Universität Augsburg

SecureMDD: Transformation of a UML application model to executable code

Nina Moebius, Marian Borek, Kurt Stenzel and Wolfgang Reif

Report 2012-11 November 2012

Institut für Informatik
D-86135 Augsburg

Abstract

The SecureMDD project provides a software engineering approach to develop secure smart card applications. The approach is model-driven and integrates formal verification to guarantee the security of the application under development. Furthermore, based on a platform-independent UML model of the application under development, the approach is able to generate executable source code for the smart cards and terminals of the application. The whole approach is fully supported by tools and all model-to-model as well as model-to-text transformations are fully implemented. This paper contains the implementation of the transformations that generate executable source code out of a platform-independent UML model of an application.

Contents

Chapter 1

Approach

The SecureMDD approach supports the development of applications that use web services, smart cards and terminals. Terminals are devices that consist of at least one smart card reader and can communicate with a smart card. Moreover a terminal usually has a user interface to communicate with the user of the application and can also call service operations. For example, a terminal could be a Home-PC or an automat.

In this Section we shortly summarize the approach. It has been described in more detail in [6]. Fig. 1.1 contains an overview.

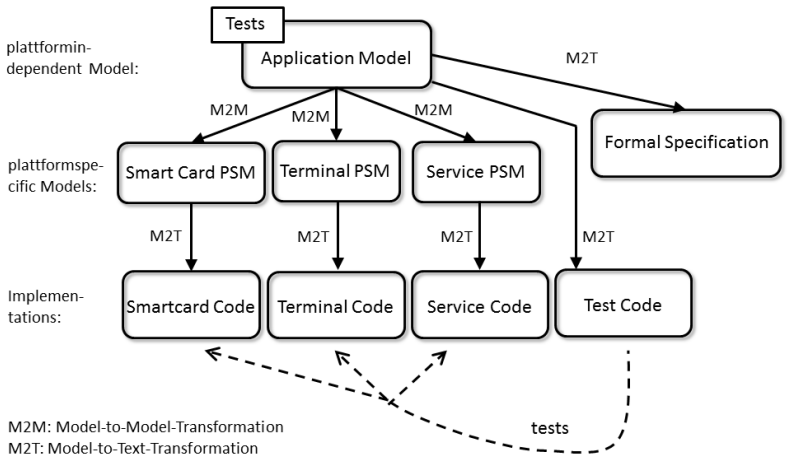

Figure 1.1: Overview of the SecureMDD approach

The development of an application starts with the creation of a platform-independent UML model. This is an abstract view of a system, omitting implementation details. To be able to model security-critical applications, UML was tailored to this domain by defining a UML profile. To support the modeling of the dynamic part of an application, i.e. the communication as well as the processing of messages, we defined a domain-specific language called Model Extension Language (MEL) which is used in UML activity diagrams. With this language it is possible to make assignments to the attributes of component classes, to create objects or to call predefined cryptographic operations. The platform-independent UML model of an application consists of all information that are needed to generate executable code as well as a formal model of the whole

application automatically.

The platform-independent model is used to generate three platform-specific models (PSM), one for the modeled smart card components, one for the terminals and one for the modeled services. The platform-specific models contain technical information about the implementation. Their aim is to minimize the gap between the platform-independent model and the generated code. For example, the Smart Card PSM contains classes with operations for the serialization and deserialization of messages. Since the communication with smart cards in the implementation is based on byte arrays (called APDUs [4]), these operations are used to transform a message into a byte array before sending it.

The platform-specific model is transformed into code using model-to-model and model-to--text transformations. For terminals and web services we generate Java code. Smart cards are programmed with Java Card ([4]), which is a subset of Java and is tailored for the use on resource-constraint devices. Thus, for smart cards we generate Java Card code.

The modeled application and especially the cryptographic protocols may contain functional and security errors. We integrated model-based testing into the approach that allows to design a test case with UML and generate test cases automatically from this model (see [5]). Then, these test cases can be executed on the generated code. Currently, tests to find protocol logic flaws as well as security tests are supported. To test the security, possible attacks are modeled with UML. The generated test cases check if these attacks are executable on the generated code.

To guarantee the security of an application, the approach integrates the formal verification of application-specific security properties (see [7]). Application-specific properties are e.g. that an electronic prescription that is stored on an electronic health card cannot be filled twice in a pharmacy. Other properties are that only genuine prescriptions can be filled, i.e. only a doctor is able to issue a prescription, and that the attacker does not get to know any prescriptions. In our opinion, for many applications application-specific security properties give better guarantees to the security of an application than standard properties like secrecy, integrity or authenticity. However, standard properties are often prerequisites for proving application-specific properties and thus, have to be verified as well. Our approach generates a formal specification based on algebraic specifications and abstract state machines (ASM) [2] for the modeled application. The generated formal model is loaded into the theorem prover KIV [1] and used for interactive verification of application-specific security properties.

The security properties that are proved to hold on the formal model also have to hold on code level. To guarantee this, the generated code has to be a refinement of the generated formal model. A solution that proves the refinement for any application that is developed with the SecureMDD approach is work in progress. A first result is the definition of a calculus for QVT (the language used to implement the model-to-model transformations) in KIV. This calculus can be used to prove the correctness of QVT transformations (see [8]).

The SecureMDD approach is fully supported by tools. For creation of the platform-independent UML model we make use of the UML modeling tool Magic Draw[1]. The transformations of the UML model into code as well as into the formal specification are executable using Eclipse plugins. The model-to-model transformations are implemented in Operational QVT[2]. The model-to-text transformations are implemented in Xpand[3]. As development environment as well as to execute the transformations we use the Eclipse Modeling Project[4]. The parsing and annotation of the MEL expressions is implemented in Java.

This paper contains the implementaion of the model-to-model- and model-to-text-transformations that generate executable Smart Card and Terminal code out of the platform-independent UML model. The paper ist structured as follows. Section 2 contains the QVT transformations that use a platform-independent UML model as source and generate a platform-specific model out of it. Section 3 describes the Xpand transformations that use the platform-specific model as source model and generate executable Java and Java Card code out of it.

[1] http://www.nomagic.com/
[2] http://www.eclipse.org/m2m/
[3] http://www.eclipse.org/modeling/m2t/?project=xpand
[4] http://www.eclipse.org/modeling/

Chapter 2

QVT transformations: Generation of a platform-specific model

This chapter contains the QVT transformations.

2.1 Prepare the platform-independent model for transformation

In this section some preliminary transformations are shown. Those prepare the platform-independent model for the generation of the platform-specific model. For example, they generate default names for the ports that do not have a name in the platform-independent model. Those transformations update the original model.

```
import cdTransformations;
import adTransformations;
import ddTransformations;
modeltype UML uses 'http://www.eclipse.org/uml2/3.0.0/UML';
library pim2psm access library cdTransformations,
        adTransformations, ddTransformations;

query preparePim(inout model:Model){
 log("-- add ActivityParameterNodes to an Partition");
  setActivityParameterNodesAndPinsToPartition(model);
 log
   ("-- set the name of represent class as partition name");
  partitionRepresentToPartitionName(model);
 log("-- create DeploymentDiagram if not present");
  createDeploymentDiagram(model);
 log("-- create DefaultPorts In DeploymentDiagramm");
  createDefaultPortsInDeploymentDiagramm(model);
return;
}
```

2.2 Transforming a platform-independent model into a Smart Card model

The transformation given in this section transform a platform-independent UML model into a platform-specific Smart Card model.

```
import cdTransformations;
import adTransformations;
import ddTransformations;
import commonQueries;
import cdQueries;
import adQueries;
import ddQueries;
import preparePim;
import swt.ModelExtensionLanguageQVTParser;
modeltype UML uses 'http://www.eclipse.org/uml2/3.0.0/UML';
transformation pim2psm
  (inout umlin : UML) access library commonQueries, cdQueries,
  adQueries, preparePim, cdTransformations, adTransformations,
  ddTransformations;
main() {
  var model : Model := umlin.objects()![Model];
  if(model.getCards().oclIsInvalid()
    ->forAll(e| e=true)) then{
   model.ownedElement.destroy();
   log("generate empty PSM");
  }else{
   model.getClassDiagram()
      .ownedComment+= object Comment{_body
        := 'Autogenerated by pim2jcpsm transformation'};
   model.ownedComment += object Comment{_body := 'Javacard'};
          model.ownedComment+=object Comment{_body :='pim2psm'};
   log("-- prepare PIM");
   preparePim(model);
   log("-- setQVTContext");
   setQVTContext(model);
    log
      ("-- clone Partitions that have abstract classes");
    clonePartition(model);
   log('changePrivateInPublicAttribute');
      changePrivateInPublicAttribute(model);
      log('removeUserStereoType');
      removeUserStereoType(model);
      log('removeAllUnusedManualClasses');
      removeAllUnusedManualClasses(model);
      log('removeAllUsages');
      removeAllUsages(model);
      log('applyTypes');
      applyTypes(model);
      log('handleLists');
      handleLists(model);
      log("getNotNecessaryElements");
      var remove : Set(Element)
        :=getNotNecessaryElements(model);
      log('generateAllInterfaces');
      generateAllInterfaces(model);
   log('generateMissingClasses');
   generateMissingClasses(model);
      log('handleAllInterfaces');
      handleAllInterfaces(model);
```

```
        log('generateCryptoClasses');
        generateHashedDataClass(model);
        generateSignedDataClass(model);
        generateEncClasses(model);
        generateMACClass(model);
        log('handleSmartCardStereotype');
        handleSmartCardStereotype(model);
        log('handleCertificateStereotype');
        handleCertificateStereotype(model);
        log('buildConstructors');
        buildConstructors(model);
        log('handleActivities');
        handleActivities(model);
        log('unapplyStereotypes');
        unapplyStereotypes(model);
    log("-- removeNotNecessaryElements");
    remove->forEach(i) { i.destroy() };
    log
      ("-- destroy elements/EObjects outside the model");
    umlin.rootObjects()
      ->select(o|o.oclIsKindOf(ActivityNode) or o
        .oclIsKindOf(ActivityEdge) or o
          .oclIsKindOf(Activity) or o
            .oclIsKindOf(Class))->forEach(n){
       umlin.removeElement(n);
      };
      }endif;
     log("Finished!");
}

helper generateMissingClasses(inout mo:Model){
 generateCommonMissingClassesPart1(mo);
 log('generateClassStore');
 generateClassStore(mo);
 log('generateClassAPDU');
 generateClassAPDU(mo);
 log('generateClassMath');
 generateClassMath(mo);
}

helper generateClassStore(inout mo:Model){
 var store: Class:=object Class{name:='Store';};
 mo.getClassDiagram().packagedElement+=store;
 insertStoreClassAttributesAndOperations(mo);
 return;
}

helper insertStoreClassAttributesAndOperations
  (inout mo:Model){
 var store: Class:=getClass(mo,'Store');
 store.ownedAttribute+=object Property{
  name:='LENGTHOFSTRING';
  type:=getDataTypePrimitive(mo,'short');
  visibility:=VisibilityKind::public;
```

```
 isLeaf:=true;
 isStatic:=true;
 _default:=getLengthOfString().toString();
};
store.ownedAttribute+=object Property{
 name:='LENGTHOFSECRET';
 type:=getDataTypePrimitive(mo,'short');
 visibility:=VisibilityKind::public;
 isLeaf:=true;
 isStatic:=true;
 _default:=getLengthOfSecret().toString();
};
store.ownedAttribute+=object Property{
 name:='LENGTHOFNONCE';
 type:=getDataTypePrimitive(mo,'short');
 visibility:=VisibilityKind::public;
 isLeaf:=true;
 isStatic:=true;
 _default:=getLengthOfNonce().toString();
};
if( hashUsed(mo)) then
store.ownedAttribute+=object Property{
 name:='COMPUTEDHASHDATALENGTH';
 type:=getDataTypePrimitive(mo,'short');
 visibility:=VisibilityKind::public;
 isLeaf:=true;
 isStatic:=true;
 _default:=computeHashDataLength().toString();
}
endif;
if (symEncUsed(mo)) then
store.ownedAttribute+=object Property{
 name:='MAXENCRYPTLENGTHSYMM';
 type:=getDataTypePrimitive(mo,'short');
 visibility:=VisibilityKind::public;
 isLeaf:=true;
 isStatic:=true;
 _default:=computeEncDataLength_Symm().toString();
}
endif;
if( asymEncUsed(mo)) then
store.ownedAttribute+=object Property{
 name:='MAXENCRYPTLENGTHASYMM';
 type:=getDataTypePrimitive(mo,'short');
 visibility:=VisibilityKind::public;
 isLeaf:=true;
 isStatic:=true;
 _default:=computeEncDataLength_Asymm().toString();
}
endif;
if( signUsed(mo)) then{
store.ownedAttribute+=object Property{
 name:='MAXSIGNEDLENGTH';
 type:=getDataTypePrimitive(mo,'short');
```

```
 visibility:=VisibilityKind::public;
 isLeaf:=true;
 isStatic:=true;
 _default:=computeSignedDataLength().toString();
};
store.ownedAttribute+=object Property{
 name:='MaxEncodingLengthSignData';
 type:=getDataTypePrimitive(mo,'short');
 visibility:=VisibilityKind::public;
 isLeaf:=true;
 isStatic:=true;
 _default
   :=computeMaxEncodingLengthSignData().toString();
}
}endif;
var codeableClasses : Set(Class)
  := mo.getCodeableClasses();
var allccs : Set(Class) := mo.getAllTargetClasses()
  ->union(codeableClasses);
var cards : Set(Class) := mo.getCards()
  ->reject(c|c.isAbstract);
cards->collect(card | if (true) then {
 var usedCodeableClasses: Set(Class):= allccs
   ->select(c| numberOfObjects(card.name, c
     .name) >= 0 );
 usedCodeableClasses->collect(c |
  store.ownedAttribute+=object Property{
   name:=c.name + 'MaxCountFor' + card.name;
   type:=getDataTypePrimitive(mo,'short');
   visibility:=VisibilityKind::public;
   isLeaf:=true;
   isStatic:=true;
   _default
     :=numberOfObjects(card.name, c.name).toString();
  }
 ); } endif);
 var usedStoreClasses: Set(Class):= cards
   ->collect(card | allccs
     ->select(c| numberOfObjects(card.name, c
       .name) >= 0 ))->flatten()->asSet();
 usedStoreClasses->collect(c |
  store.ownedOperation+=object Operation{
    name:='new'+c.name;
    type:=c;
    visibility:=VisibilityKind::public;
    isStatic:=true;
   }
 );
 usedStoreClasses->collect(c |
  if(c.ownedAttribute->size() > 0) then{
    var o: Operation:= object Operation{
     name:='new'+c.name;
     type:=c;
     visibility:=VisibilityKind::public;
```

```
      isStatic:=true;
     };
     c.ownedAttribute->collect(a |
      o.ownedParameter+= object Parameter{
       name:=a.name;
       type:=a.type;
       upper:=a.upper;
       lower:=a.lower;
      }
     );
     store.ownedOperation+=o
   }endif
  );
 store.ownedOperation+=object Operation{
  name:='initAll';
  visibility:=VisibilityKind::public;
  isStatic:=true;
 };
 store.ownedOperation+=object Operation{
  name:='reset';
  visibility:=VisibilityKind::public;
  isStatic:=true;
 };
 return;
}

helper generateClassAPDU(inout mo : Model)
{
 var c: Class:= object Class
 {
  name:= 'APDU';
  visibility := VisibilityKind::public;
 };
 mo.getClassDiagram().packagedElement+=c;
 return;
}

helper generateClassMath(inout mo:Model)
{
 var c:Class:=object Class
 {
  name:='Math';
  visibility:=VisibilityKind::public;
 };
 mo.getClassDiagram().packagedElement+=c;
 insertDefaultMathOperation(mo);
 return;
}

helper insertDefaultMathOperation(inout mo:Model){
var source: Class:=mo.getClassDiagramClasses()
  ->any(c|c.name='Math');
 source.ownedOperation+=object Operation{
  name:='plus';
```

```
 visibility:=VisibilityKind::public;
 isStatic:=true;
 type:=getDataTypePrimitive(mo, 'short');
 var m1 : Parameter := object Parameter{name
   := 'x'; type := getDataTypePrimitive(mo,'short');};
 var m2 : Parameter := object Parameter{name
   := 'y'; type := getDataTypePrimitive(mo,'short');};
 ownedParameter+=m1;
 ownedParameter+=m2;
};
source.ownedOperation+=object Operation{
 name:='overflowsPlus';
 visibility:=VisibilityKind::public;
 isStatic:=true;
 type:= getDataTypePrimitive(mo, 'boolean');
 var m3 : Parameter := object Parameter{name
   := 'x'; type := getDataTypePrimitive(mo,'short');};
 var m4 : Parameter := object Parameter{name
   := 'y'; type := getDataTypePrimitive(mo,'short');};
 ownedParameter+=m3;
 ownedParameter+=m4;
};
source.ownedOperation+=object Operation{
 name:='minus';
 visibility:=VisibilityKind::public;
 isStatic:=true;
 type:= getDataTypePrimitive(mo, 'short');
 var m5 : Parameter := object Parameter{name
   := 'x'; type := getDataTypePrimitive(mo,'short');};
 var m6 : Parameter := object Parameter{name
   := 'y'; type := getDataTypePrimitive(mo,'short');};
 ownedParameter+=m5;
 ownedParameter+=m6;
};
source.ownedOperation+=object Operation{
 name:='div';
 visibility:=VisibilityKind::public;
 isStatic:=true;
 type:= getDataTypePrimitive(mo, 'short');
 var m7 : Parameter := object Parameter{name
   := 'x'; type := getDataTypePrimitive(mo,'short');};
 var m8 : Parameter := object Parameter{name
   := 'y'; type := getDataTypePrimitive(mo,'short');};
 ownedParameter+=m7;
 ownedParameter+=m8;
};
source.ownedOperation+=object Operation{
 name:='rem';
 visibility:=VisibilityKind::public;
 isStatic:=true;
 type:= getDataTypePrimitive(mo, 'short');
 var m9 : Parameter := object Parameter{name
   := 'x'; type := getDataTypePrimitive(mo,'short');};
 var m10 : Parameter := object Parameter{name
```

```
    := 'y'; type := getDataTypePrimitive(mo,'short');};
  ownedParameter+=m9;
  ownedParameter+=m10;
 };
 source.ownedOperation+=object Operation{
  name:='mult';
  visibility:=VisibilityKind::public;
  isStatic:=true;
  type:= getDataTypePrimitive(mo, 'short');
  var m11 : Parameter := object Parameter{name
    := 'x'; type := getDataTypePrimitive(mo,'short');};
  var m12 : Parameter := object Parameter{name
    := 'y'; type := getDataTypePrimitive(mo,'short');};
  ownedParameter+=m11;
  ownedParameter+=m12;
 };
return;
}

helper insertDefaultSimpleCommOperations
  (inout mo:Model){
var source: Class:=mo.getClassDiagramClasses()
  ->any(c|c.name='SimpleComm');
source.ownedOperation+=object Operation{
  name:='resetGate';
  visibility:=VisibilityKind::public;};
 var gate:Class:=mo.getClassDiagramClasses()
   ->any(c|c.name='Gate')->any(true);
 source.ownedOperation+=object Operation{
  name:='setGate';
  visibility:=VisibilityKind::public;
  var s1 : Parameter := object Parameter{name
    := 'gate'; type := gate;};
  ownedParameter+=s1;
  };
 var apdu:Class:=mo.getClassDiagramClasses()
   ->any(c|c.name='APDU')->any(true);
 source.ownedOperation+=object Operation{
  name:='sendAPDU';
  visibility:=VisibilityKind::public;
  var s2 : Parameter := object Parameter{name
    := 'apdu'; type := apdu;};
  var s3 : Parameter := object Parameter{name
    := 'data'; type
      := getDataTypePrimitive(mo,'byte');upper
        :=-1;lower:=0;};
  var s4 : Parameter := object Parameter{name
    := 'rem_len'; type
      := getDataTypePrimitive(mo,'short');};
  var s5 : Parameter := object Parameter{name
    := 'first'; type
      := getDataTypePrimitive(mo,'boolean');};
  ownedParameter+=s2;
  ownedParameter+=s3;
```

```
  ownedParameter+=s4;
  ownedParameter+=s5;};
 source.ownedOperation+=object Operation{
  name:='appletInitialization';
  visibility:=VisibilityKind::public;
  isAbstract:=true;
  var s6 : Parameter := object Parameter{name
    := 'buf'; type
      := getDataTypePrimitive(mo,'byte');upper
        :=-1;lower:=0;};
  var s7 : Parameter := object Parameter{name
    := 'offset'; type
      := getDataTypePrimitive(mo,'short');};
  ownedParameter+=s6;
  ownedParameter+=s7;};
 source.ownedOperation+=object Operation{
  name:='install';
  visibility:=VisibilityKind::public;
  isStatic:=true;
  var s8 : Parameter := object Parameter{name
    := 'bArray'; type
      := getDataTypePrimitive(mo,'byte');upper
        :=-1;lower:=0;};
  var s9 : Parameter := object Parameter{name
    := 'offset'; type
      := getDataTypePrimitive(mo,'short');};
  var s10 : Parameter := object Parameter{name
    := 'bLength'; type
      := getDataTypePrimitive(mo,'byte');};
  ownedParameter+=s8;
  ownedParameter+=s9;
  ownedParameter+=s10;};
 source.ownedOperation+=object Operation{
  name:='stop';
  isStatic:=true;
  };
return;
}

helper changePrivateInPublicAttribute(inout mo: Model){
 mo.getClassDiagramClasses()->collect(c|
  c.ownedAttribute
    ->collect(a| a.visibility:= VisibilityKind::public)
 );
 return;
}

helper addStereotypeSingletonToCodingclass
  (inout mo : Model)
{
 var source : Class := mo.getClassDiagramClasses()
   ->any(c | c.name = 'Coding');
 var st : Stereotype := mo.getApplicableStereotypes()
   ->any(st | st.name = 'Singleton');
```

```
 if (st = null) then logging
   ("addStereotypeSingletonToCodingclass:
     Stereotype is null!") endif;
 source.applyStereotype(st);
 return;
}

helper handleSmartCardStereotype(inout mo : Model)
{
 var cd : Package := mo.getClassDiagram();
 var smartcards : Set(Class) := mo.getCards();
 var codingClass : Class
   := mo.getClassDiagramClasses()
     ->any(c | c.name = 'Coding').oclAsType(Class);
 var msgClass : Class := mo.getClassDiagramClasses()
   ->any(c | c
     .getAppliedStereotype
       ('SecureMDD::Message')<>null);
 var apduClass : Class := mo.getClassDiagramClasses()
   ->any(c | c.name = 'APDU');
 var appletClass : Class := object Class{name
   :='javacard.framework.Applet';};
 var simpleCommClass : Class := object Class{name
   := 'SimpleComm'; isAbstract:=true;};
 cd.packagedElement += simpleCommClass;
 mo.getCards()->collect(card|
  if(card.general->isEmpty()) then {
   var scGerneralization : Generalization
     := object Generalization{general
       := simpleCommClass; specific := card;};
   var simpleCommGerneralization : Generalization
     := object Generalization{general
       := appletClass; specific := simpleCommClass;};
  }endif
 );
 var simpleCommAssociation : Association
   := object Association{};
 var codingEnd : Property := object Property{name
   := 'coding'; visibility
     := VisibilityKind::protected; association
       := simpleCommAssociation; aggregation
         := AggregationKind::none; lower := 1; upper
           := 1; type := codingClass; isStatic
             := true;};
 var scEnd : Property := object Property{visibility
   := VisibilityKind::private; association
     := simpleCommAssociation; aggregation
       := AggregationKind::none; type
         := simpleCommClass};
 simpleCommClass
   .ownedOperation += generateSimpleCommOperations
     (msgClass, apduClass);
 mo.getCards()->collect(card|
  card.ownedOperation += object Operation{name
```

```
    := 'process'; ownedParameter += object Parameter{name
      :='msg'; type:=msgClass;};}
 );
 simpleCommClass.ownedAttribute += codingEnd;
 generateSimpleCommAttributes(mo);
 simpleCommAssociation.ownedEnd += scEnd;
 cd.packagedElement += appletClass;
 cd.packagedElement += simpleCommAssociation;
 return;
}

helper generateSimpleCommOperations
  (inout msg : Class, inout apdu : Class) : Set
    (Operation)
{
 var mo : Model := umlin.objects()![Model];
 var gate: Class:=mo.getClassDiagramClasses()
  ->any(c|c.name='Gate');
 var newOperations : Set(Operation)
  := msg.ownedOperation->select(e | false);
 newOperations += object Operation{name
  := 'process'; ownedParameter += object Parameter{name
    :='apdu'; type:=apdu;};};
 newOperations += object Operation{name
  := 'process'; isAbstract
    :=true;ownedParameter += object Parameter{name
      :='msg'; type:=msg;};};
 newOperations += object Operation{name
  := 'sendMsg'; isStatic:=true; visibility
    :=VisibilityKind::protected; ownedParameter += object Parameter{name
      :='msg'; type:=msg;};};
 newOperations += object Operation{name
  := 'resetGate'; visibility
    :=VisibilityKind::public;};
 newOperations += object Operation{name
  := 'setGate'; visibility
    :=VisibilityKind::public; ownedParameter += object Parameter{name
      :='gate'; type:=gate;};};
 newOperations += object Operation{name
  := 'sendAPDU'; visibility:=VisibilityKind::public;
 ownedParameter += object Parameter{name
  :='apdu'; type:=apdu;};
 ownedParameter += object Parameter{name
  :='data'; type
    :=getDataTypePrimitive(mo,'byte'); upper
      :=-1; lower:=0;};
 ownedParameter += object Parameter{name
  :='rem_len'; type
    :=getDataTypePrimitive(mo,'short');};
 ownedParameter += object Parameter{name
  :='first'; type
    :=getDataTypePrimitive(mo,'boolean');};};
 newOperations += object Operation{name
  := 'appletInitialization'; isAbstract
```

```
      :=true; visibility:=VisibilityKind::public;
  ownedParameter += object Parameter{name:='buf'; type
    :=getDataTypePrimitive(mo,'byte'); upper:=-1;lower
      :=0;};
  ownedParameter += object Parameter{name
    :='offset'; type
      :=getDataTypePrimitive(mo,'short');};};
  newOperations += object Operation{name
    := 'install'; isStatic:=true; visibility
      :=VisibilityKind::public;
  ownedParameter += object Parameter{name
    :='bArray'; type
      :=getDataTypePrimitive(mo,'byte');upper
        :=-1;lower:=0;};
  ownedParameter += object Parameter{name
    :='bOffset'; type
      :=getDataTypePrimitive(mo,'short');};
  ownedParameter += object Parameter{name
    :='bLength'; type
      :=getDataTypePrimitive(mo,'byte');};};
  newOperations += object Operation{name
    := 'stop';isStatic:=true;};
  return newOperations;
}

helper generateSimpleCommAttributes(inout mo: Model) {
  var source : Class := mo.getClassDiagramClasses()
    ->any(c | c.name = 'SimpleComm');
  source.ownedAttribute+=object Property{
   name:='INIT_INSTRUCTION';
   type:=getDataTypePrimitive(mo,'byte');
   visibility:=VisibilityKind::public;
   isLeaf:=true;
   isStatic:=true;
   _default:="4";
  };
  source.ownedAttribute+=object Property{
   name:='RESUME_INSTRUCTION';
   type:=getDataTypePrimitive(mo,'byte');
   visibility:=VisibilityKind::public;
   isLeaf:=true;
   isStatic:=true;
   _default:="6";
  };
  source.ownedAttribute+=object Property{
   name:='ENCBUF_MAXLEN';
   type:=getDataTypePrimitive(mo,'short');
   visibility:=VisibilityKind::public;
   isLeaf:=true;
   isStatic:=true;
   _default
     :=computeMaxEncodingLengthMessages().toString();
  };
  source.ownedAttribute+=object Property{
```

```
  name:='DECBUF_MAXLEN';
  type:=getDataTypePrimitive(mo,'short');
  visibility:=VisibilityKind::public;
  isLeaf:=true;
  isStatic:=true;
  _default
    :=computeMaxEncodingLengthMessages().toString();
 };
 source.ownedAttribute+=object Property{
  name:='APDU_MAXLEN';
  type:=getDataTypePrimitive(mo,'short');
  visibility:=VisibilityKind::public;
  isLeaf:=true;
  isStatic:=true;
  _default:="255";
 };
 source.ownedAttribute+=object Property{
  name:='OUTMSG_MAXLEN';
  type:=getDataTypePrimitive(mo,'short');
  visibility:=VisibilityKind::public;
  isLeaf:=true;
  isStatic:=true;
  _default:="254";
 };
 return;
}

helper generateDefaultByteValues4Constants
  (inout source: OrderedSet
    (Property), dValue : Integer) : OrderedSet
      (Property)
{
 var params:OrderedSet(Property);
 var value:Integer:=dValue;
 source->collect(p|
  if(true) then{
   params+=object Property{
    name:=p.name;
    visibility:=VisibilityKind::public;
    isStatic:=true;
    isLeaf:=true;
    type :=getDataTypePrimitive(p.getModel(), 'short');
    defaultValue:=object OpaqueExpression{_body
      :=value.toString()};
   };
   value:=value+1;
  }endif
 );
 return params;
}
```

2.3 Transforming a platform-independent model into a Terminal model

The transformation given in this section transform a platform-independent UML model into a platform-specific Terminal model.

```
import cdTransformations;
import adTransformations;
import ddTransformations;
import commonQueries;
import cdQueries;
import adQueries;
import ddQueries;
import preparePim;
import swt.ModelExtensionLanguageQVTParser;
modeltype UML uses 'http://www.eclipse.org/uml2/3.0.0/UML';
transformation pim2psm
  (inout umlin : UML) access library commonQueries, cdQueries,
   adQueries, preparePim, cdTransformations, adTransformations,
   ddTransformations;
main() {
  var model : Model
    := umlin
      .rootObjects()![Model]; // getUMLModel
        (); //umlin.rootObjects()![Model];
   model.getClassDiagram()
     .ownedComment+= object Comment{_body
       := 'Autogenerated by pim2tpsm transformation'};
   model.ownedComment += object Comment{_body
     := 'Terminal'};
        model.ownedComment+=object Comment{_body
          :='pim2psm'};
  log("-- prepare PIM");
  preparePim(model);
  log("-- setQVTContext");
  setQVTContext(model);
  log
    ("-- clone Partitions that have abstract classes");
  clonePartition(model);
    log("-- handle UsermassageIsMessage");
    handleUsermessageIsMessage(model);
    log("--remove all unused Manual Classes");
    removeAllUnusedManualClasses(model);
    log("-- remove all usages");
    removeAllUsages(model);
    log("-- change PrimitiveTypes");
    applyTypes(model);
    log("-- handle list objects");
    handleLists(model);
    logging("-- getNotNecessaryElements");
    var remove : Set(Element)
      :=getNotNecessaryElements(model);
    var cds : Set(Class)
      := model.getClassDiagramClasses();
```

```
      log("-- generate all Interfaces");
      generateAllInterfaces(model);
      log
      ("-- generate all classes that are not in the pim");
      generateMissingClasses(model);
      log("-- associate classe who implements
            interfaces with that interfaces");
      handleAllInterfaces(model);
      log("-- modify hasheddata class");
      generateHashedDataClass(model);
      log("-- modify signeddata class");
      generateSignedDataClass(model);
      log("-- modify encryption classes");
      generateEncClasses(model);
      log("-- generate MACdata class");
      generateMACClass(model);
      log("-- manage terminal stereotype");
      handleTerminalStereotype(model);
      logging("-- handleCertificateStereotype");
      handleCertificateStereotype(model);
      logging("-- add constructors to the transitive
           closure of the message and the sc class");
      buildConstructors(model);
      logging("-- handle activities");
      handleActivities(model);
      log("-- unapply data stereotypes");
      unapplyStereotypes(model);
      log("-- removeNotNecessaryElements");
      remove->forEach(i) { i.destroy() };
      log("-- destroy elements/EObjects outside the model");
      umlin.rootObjects()
      ->select(o|o.oclIsKindOf(ActivityNode) or o
       .oclIsKindOf(ActivityEdge) or o
         .oclIsKindOf(Activity) or o
           .oclIsKindOf(Class))->forEach(n){
      umlin.removeElement(n);
  };
      log("Finished!");
}

query generateMissingClasses(inout mo:Model){
 generateCommonMissingClassesPart1(mo);
 generateClassSWTReader(mo);
 generateClassUserinterface(mo);
 generateCommonMissingClassesPart2(mo);
return;
}

query writeInitialServiceConnectionInActivities
  (inout mo: Model, activities : Set
    (Activity), servers: Set(Class)){
 var terminals : Set(Class):=mo.getTerminals();
 var dependencySupplier : Set(AcceptEventAction)
   :=mo.getActivityDiagram().ownedElement[Dependency]
```

```
    .supplier[AcceptEventAction]->asSet();
var allTerminalAcceptNodes: Set(AcceptEventAction)
  := getAllTargetAcceptNodes(mo);
activities->collect(a|
 if(true) then{
  var terminalPartitions: Set(ActivityPartition)
    :=a.partition->select(p|terminals.name
      ->includes(p.getClassName()));
  var serverPartitions: Set(ActivityPartition)
    := a.partition->select(p|
   servers.name->includes(p.getClassName()));
  if(serverPartitions->notEmpty())then{
   var allControlFlows: Set(ControlFlow)
     := a.ownedElement[ControlFlow]->asSet();
   terminalPartitions->collect(t|
    if(true) then {
     var allFirstTerminalAcceptNodes: Set
       (AcceptEventAction):= allTerminalAcceptNodes
         ->select(aNode|
      aNode.owner[Activity]->any(true)=a
      and not dependencySupplier->includes(aNode)
      and aNode.inPartition->any(true)=t
     );
     if(allFirstTerminalAcceptNodes->size() = 1) then{
      var firstTerminalAcceptNode: AcceptEventAction
        :=allFirstTerminalAcceptNodes
          ->any(true).oclAsType(AcceptEventAction);
      var actionName: String:="";
      serverPartitions->collect(s|
       if(allControlFlows
         ->exists(c|c.source.inPartition
           ->any(true)=t and c.target.inPartition
             ->any(true)=s)) then{
        if(not isServerStateful(mo,s
          .getClassName()) ) then{
         if(getClass(mo,s.getClassName())
           .isAbstract) then{
          getClass(mo,s.getClassName())
            .getSubclasses()->collect(subclass|
           if(true) then{
            actionName
              :=actionName+"initializeServiceConnection
                ("+getNode(mo,t.getClassName())
                  .getAllAttributes()
             ->any(att|att.type[Node]
               ->any(true).name=subclass.name)
                 .name+");\n";
           }endif
          );
         }else{
          var port:String
            :=getNode(mo,t.getClassName())
              .getAllAttributes()
           ->any(att|att.type[Node]
```

```
                ->any(true).name=s.getClassName()).name;
              actionName
                :=actionName+"initializeServiceConnection
                  ("+port+");\n";
             }endif;
            }endif
           }endif
          );
          if(actionName<>"")then{
           log
             ("actionName "+actionName+" for Terminal "+t
               .name+" in activity: "+a.name);
           var action: CallBehaviorAction
             := object CallBehaviorAction{
            name:=actionName;
            outgoing:=firstTerminalAcceptNode.outgoing;
            firstTerminalAcceptNode.outgoing:=null;
           };
           var edge: ActivityEdge:=object ControlFlow{
            source:=firstTerminalAcceptNode;
            target:=action;
            weight:= object LiteralInteger{value:=1};
           };
           edge.inPartition+=firstTerminalAcceptNode
             .inPartition;
           action.inPartition+=firstTerminalAcceptNode
             .inPartition;
           getActivity(firstTerminalAcceptNode)
             .edge+=edge;
           getActivity(firstTerminalAcceptNode)
             .node+=action;
          }endif;
         }else{
          log("activity: "+a
            .name+" ist eine Methode ohne send
            und receive oder es gibt einen Fehler bei den Dependencies");
         }endif
        }endif
       );
     }endif
    }endif
   );
return;
}

query generateClassUserinterface(inout mo: Model)
{
 var userinterface : Interface
   := object Interface{ name := 'Userinterface';};
 mo.getClassDiagram().packagedElement+=userinterface;
 return
}
 var activities : Set(Activity):= mo.getActivities();
 var activityDiagramm: Package
```

```
   := mo.getActivityDiagram();
 var nothingFound=true;
 activities->collect(activity|
  if(true) then{
   var partitions: Set(ActivityPartition)
     := activity.partition;
   var upperClassPartition: ActivityPartition
     := partitions
       ->any(p|getClass(mo,p.getClassName())
         .isAbstract);
   if(not upperClassPartition.oclIsUndefined()) then{
    nothingFound:=false;
    var subClasses: Set(Class)
     :=getClass(mo,upperClassPartition
       .getClassName()).getSubclasses();
    subClasses->collect(sc|
     if(true) then {
      var a: Activity
       := activity.deepclone().oclAsType(Activity);
      a.partition
       ->any(p| p.name=upperClassPartition.name)
         .name:=sc.name;
      activityDiagramm.packagedElement+=a;
     }endif
    );
    activity.destroy();
   }endif;
  }endif
 );
 if(nothingFound = false) then{
  clonePartition(mo);
 }endif;
 return;

query handleUsermessageIsMessage(inout mo:Model){
  var message: Class:=mo.getClassDiagramClasses()
    ->select(c|c.hasStereotype("SecureMDD::Message"))
      ->any(true);
  var userMessages : Set(Class)
    :=mo.getClassDiagramClasses()
      ->select(c|c
        .hasStereotype("SecureMDD::Usermessage"));
  userMessages->collect(um | um.general+=message);
 return;
}

query clearServers(mo:Model){
 var usedServers: Set(Class) :=object Set(Class){};
 mo.getTerminals()
   ->collect(t|usedServers:=usedServers
     ->union(getAllDirectlyUsedServer(t)));
 usedServers->collect(s|
  if(true)then{
   s.ownedAttribute
```

```
      ->select(a|not a.oclIsTypeOf(Association))
        ->collect(i |umlin.removeElement(i));
    s.ownedOperation
      ->collect(i |umlin.removeElement(i));
  }endif
 );
 return;
}

query removeAllUsages(inout mo : Model){
 var terminals : Set(Class) := mo.getTerminals();
 var smartcards : Set(Class) := mo.getCards();
 var cd := mo.getClassDiagram();
 var usages : Bag(Usage) := cd.ownedElement
   ->select(e | e.oclIsTypeOf(Usage)).oclAsType(Usage);
 usages->destroy();
 return;
}

query getAllTerminalAcceptNodes(in mo : Model) : Set
  (AcceptEventAction){
 var nodes : Set(AcceptEventAction)
   := mo.getActivities()
     .ownedElement[AcceptEventAction]
 ->select(e|e.isInTerminalPartition())->asSet();
 return nodes;
}

query getAllTerminalSendNodes(in mo : Model) : Set
  (SendSignalAction){
 var nodes : Set(SendSignalAction)
   := mo.getActivities().ownedElement[SendSignalAction]
 ->select(e|e.isInTerminalPartition())->asSet();
 return nodes;
}

query ActivityNode::isInTerminalPartition() : Boolean {
 return self.getModel().getTerminals()
   ->exists(t|t.name=self.inPartition
     ->any(true).getClassName());
}

query setReceiveString
  (inout node : ActivityNode, a : ActivityNode, an :Set
    (ActivityNode), servers: Set(Class)){
 log("setReceiveString on node " + node
   .name + " with cur a: " + a.name);
 if(not an->includes(a) or a.outgoing
   ->isEmpty()) then return endif;
 var ret : String="";
 if(not a.outgoing.target[AcceptEventAction]
   ->isEmpty()) then {
  var rec : String
    := getSendReceiveClass(a.outgoing
```

```
      .target[AcceptEventAction]->any(true).name);
  var targetPartitionName: String:= node.outgoing
    ->any(true).target.inPartition
      ->any(true).getClassName();
  if(servers
    ->exists(s|s.name=targetPartitionName) ) then{
   ret
     := "\nprocess"+rec.repr()
       .toFirstUpper() + "(ret_o);";
  }
  else{
   ret:= "\nCoding.getInstance().decodeInit();\n" +
        rec + " ret_o = (" + rec + ") Coding
          .getInstance().decode(ret);\n" +
        "process"+rec.repr()
          .toFirstUpper() + "(ret_o);";
  }endif;
  node.name := node.name + ret;
  return;
 }else{
  a.outgoing.target->collect(t|
   setReceiveString(node, t, an->excluding(a),servers)
  );
 }endif;
}

query handleTerminalStereotype(inout mo : Model)
{
 var cd : Package := mo.getClassDiagram();
 var terminals : Set(Class) := mo.getTerminals();
 var codingClass : Class
   := mo.getClassDiagramClasses()
     ->any(c | c.name = 'Coding').oclAsType(Class);
 var msgClass : Class := mo.getClassDiagramClasses()
   ->any(c | c
     .getAppliedStereotype
       ('SecureMDD::Message')<>null);
 terminals->reject(e|e.isAbstract)
   ->collect(t| mo.getCards()->reject(e|e.isAbstract)
     ->collect(c|t
       .ownedOperation+=generateCardInitializeMethod
         (mo, c)) );
 terminals->reject(e|e.isAbstract)->collect(t|
  t.ownedOperation += object Operation{
   name := 'initReader';
   ownedParameter += object Parameter{
    name := 'reader';
    type := getClass(mo, 'SWTReader');
   };
   ownedParameter += object Parameter{
    name := 'port';
    type := getDataTypePrimitive(mo, 'int');
   };
  }
```

```
);
terminals->reject(e|e.isAbstract)->collect(t|
 t.ownedOperation += object Operation{
  name := 'setUserinterface';
  ownedParameter += object Parameter{
   name := 'u';
   type := getClass(mo, 'Userinterface');
  };
 }
);
terminals->reject(e|e.isAbstract)->collect(t|
 t.ownedOperation += object Operation{
  name := 'process';
  ownedParameter += object Parameter{
   name := 'msg';
   type := mo.getClassDiagramClasses()
     ->any(e | e.hasStereotype("SecureMDD::Message"));
  };
 }
);
terminals->reject(e|e.isAbstract)->collect(t|
 t.ownedOperation += object Operation{
  name := 'sendMSG';
  ownedParameter += object Parameter{
   name := 'msg';
   type := mo.getClassDiagramClasses()
     ->any(e | e.hasStereotype("SecureMDD::Message"));
  };
 }
);
terminals->reject(e|e.isAbstract)->collect(t|
 t.ownedOperation += object Operation{
  name := 'sendMSG';
  ownedParameter += object Parameter{
   name := 'msg';
   type := mo.getClassDiagramClasses()
     ->any(e | e.hasStereotype("SecureMDD::Message"));
  };
  ownedParameter += object Parameter{
   name := 'port';
   type := getDataTypePrimitive(mo, 'int');
  };
 }
);
nTerminals->collect(term|
 if(true) then {
  var invokedStatefulServices : Set(Property)
    :=term.getAllAttributes()
      ->select(att|att.type[Node]
        ->any(true)
          .hasStereotype
            ("SecureMDD::Service") and att
              .type[Node]
                ->any(true).isStatefulServer());
```

```
    if(invokedStatefulServices->notEmpty()) then{
     var op: Operation:= object Operation{
      name := "initializeServiceConnection";
      visibility:= VisibilityKind::private;
      ownedParameter += object Parameter{
       name := 'port';
       type := getDataTypePrimitive(mo,'int');
      };
     };
     mo.getTerminals()
       ->any(t|t.name=term.name).ownedOperation +=op;
    }endif;
   }endif
  return;
}

query generateConstructorForTerminal
   (inout source: Class) {
  log("generating constructor for terminal " + source
    .name);
  var ops : OrderedSet(Operation)
    := object OrderedSet(Operation){};
  ops += object Operation{name:= source.name};
  var attrs : OrderedSet(Property)
    := source.getAllAttributesForInitialization()
      ->asOrderedSet()->sortedBy(name);
  if(not attrs->isEmpty()) then {
   var res : Operation := object Operation{name
     := source.name};
   attrs->collect(att|
    if(true) then{
     res.ownedParameter +=
     object Parameter{name:=att.name; type
       :=att.type.translateType();};
    }endif
   );
   ops += res;
  } endif;
  source.ownedOperation+=ops;
  return;
}

query generateClassPorts(in mo: Model)
{
  var portsc : Class:= object Class{ name := 'Ports';};
  mo.getClassDiagram().packagedElement+=portsc;
  var numberType : PrimitiveType
    :=  getDataTypePrimitive(mo, 'int');
  var num:Integer:=0;
  mo.getAllPorts()
    ->collect(p|if(p.name.oclIsUndefined() or p.name
      .match('')) then {
   p.name
     := p.owner.oclAsType(Node).name.toFirstUpper()
```

```
    + '2'
    + p.type.name.toFirstUpper()
    + '_default_'
    + num.toString();
   num := num+1
  } endif);
  var ports : OrderedSet(Property)
    := mo.getTerminal2CardPorts();
  ports->collect(p| if(true) then {
   portsc.ownedAttribute += object Property{
    name:=p.name;
    visibility := VisibilityKind::public;
    _default := (ports->indexOf(p)-1).toString();
    type := numberType;
    isLeaf := true;
    isStatic := true;}
   } endif
  );
  ports := mo.getTerminal2UserPorts();
  ports->collect(p|
   portsc.ownedAttribute += object Property{
    name:=p.name;
    visibility := VisibilityKind::public;
    _default := (ports->indexOf(p)-1).toString();
    type := numberType;
    isLeaf := true;
    isStatic := true;}
  );
  ports := mo.getTermServ2ServerPorts();
   ports->collect(p| if(true) then {
    portsc.ownedAttribute += object Property{
     name:=p.name;
     visibility := VisibilityKind::public;
     _default := (ports->indexOf(p)-1).toString();
     type := numberType;
     isLeaf := true;
     isStatic := true;}
    } endif
   );
  return
}

query Model::getTerminal2UserPorts() : OrderedSet
  (Property){
   return self.getAllPorts()
     ->select(p|p.owner
        .hasStereotype('SecureMDD::Terminal')
    and (p.type.hasStereotype("SecureMDD::User") or
    (
     not p.type.hasStereotype("SecureMDD::Smartcard")
     and not p.type
        .hasStereotype("SecureMDD::Terminal")
     and not p.type.hasStereotype("SecureMDD::Service")
  )
```

```
  )
   )->asOrderedSet();
}

query Model::getTermServ2ServerPorts() : OrderedSet
  (Property){
 return self.getAllPorts()->select(p|p.isNavigable()
 and p.type.hasStereotype("SecureMDD::Service"))
   ->asOrderedSet();
}

query Model::getTerminal2CardPorts
  (terminal : String) : OrderedSet(Property){
 return self.getAllPorts()->select(p|p.owner[Node]
   ->any(true).name=terminal
  and p.type.hasStereotype("SecureMDD::Smartcard"))
    ->asOrderedSet()->sortedBy(name);

query isServerStateful(server : Class) : Boolean{
 if(not server.isAbstract) then{
  if(server.getValue(server
    .getAppliedStereotype
      ("SecureMDD::Service"),"stateful")
        .oclAsType(Boolean)) then{
   return true
  }endif;
   if(server.getGeneralServer().getValue(server
     .getGeneralServer()
       .getAppliedStereotype
         ("SecureMDD::Service"),"stateful")
           .oclAsType(Boolean)) then{
   return true
  }endif;
 }endif;
 return false;
}

query isServerStateful
  (mo: Model, server : String) : Boolean{
  var node:Node
    :=mo.getDeploymentDiagram().ownedElement[Node]
      ->any(n|n.name=server);
  if(node.getValue(node
    .getAppliedStereotype
      ("SecureMDD::Service"),"stateful")
        .oclAsType(Boolean)) then{
   return true
  }endif;
 return false;
}

query Model::getAllStatefulServer(): Set(Class){
 return self.getServers()
   ->select(s|isServerStateful(s));
}
```

```
query Model::getAllDirectlyUsedStatefulServer
  (terminal :Class): Set(Class){
 return getAllDirectlyUsedServer(terminal)
   ->select(s|isServerStateful(s));
}

query generateAllDirectlyUsedServerManagerNodes
  (mo : Model){
 mo.getTerminals()->collect(t|
 if(true)then{
  var allDirectlyUsedStatefulServer:Set(Class)
    :=mo.getAllDirectlyUsedStatefulServer(t);
  allDirectlyUsedStatefulServer->collect(s|
   if(true) then {
    var node: Node:= object Node {name
      := s.name+'StatefulManager';};
    mo.getDeploymentDiagram().packagedElement += node;
    node.applyStereotype(node
      .getApplicableStereotype("SecureMDD::Service"));
    var sNode: Node:=getNode(s);
    var incommingNodes: Set(Node)
      :=getIncommingNodes(sNode);
    incommingNodes->collect(n|
     if(true)then{
      var comPath: CommunicationPath
        := object CommunicationPath{
       var client: Property:=object Property{type
         :=n; visibility:=VisibilityKind::private; };
       var supplier: Property:=object Property{type
         :=getNode(node); visibility
           :=VisibilityKind::private;};
       ownedEnd+=client;
       ownedEnd+=supplier;
      };
      mo.getDeploymentDiagram()
        .packagedElement +=comPath;
      n.ownedAttribute+=comPath.memberEnd
        ->any(e|e.type.name=getNode(node).name);
     }endif
    );
   }endif
  );
 }endif
 );
 return;
}

query generateAssociationsFromTerminalsToDirectlyUsedServers
  (mo : Model){
 mo.getTerminals()->collect(t|
 if(true)then{
  var allDirectlyUsedServer : Set(Class)
    :=getAllDirectlyUsedServer(t);
  allDirectlyUsedServer->collect(s|
```

```
    if(true)then{
     var ass: Association:= object Association{
      var client: Property:=object Property{type
        :=t; visibility:=VisibilityKind::private; };
      var supplier: Property:=object Property{name
        :=s.name.toFirstLower();type:=s;visibility
          :=VisibilityKind::private;};
      ownedEnd+=client;
      ownedEnd+=supplier;
     };
     mo.getClassDiagram().packagedElement+=ass;
     t.ownedAttribute+=ass.memberEnd
       ->any(e|e.type.name=s.name);
    }endif
   );
  }endif
  );
return;
}

query getAllDirectlyUsedServer(terminal : Class) : Set
   (Class){
  var outgoingNodes: Set(Node)
    :=getOutgoingNodes(getNode(terminal));
  return terminal.getModel().getServers()
    ->select(c|outgoingNodes->exists(n|n.name=c.name));
}

query getNextAttribute(inout attrs_ : Set
   (Property)) : Property {
  var att : Property := null;
  var attrs : Set(Property) := attrs_;
  while (not attrs->isEmpty()){
   var att_temp : Property := attrs
     ->any(true).oclAsType(Property);
   if(att_temp.name < att.name) then {
    att := att_temp;
   }endif;
   attrs := attrs->excluding(att_temp);
  } ;
  var attrs_temp :Set(Property)
    := object Set(Property){};
  attrs_temp += att;
  attrs_temp->sortedBy(name);
  return att;
}

query tcAlgorythm(in cd : Package, cRes : Bag
   (Class), cClass : Set(Class)) : Bag(Class)
{
  var res : Bag(Class) := cRes;
  var temp : Bag(Class) := cd.allInstances(Class)
    ->select(c | c.allInstances(Generalization)
      ->exists(g | cClass->includes(g.general
```

```
      ->oclAsType(Class)->any(true)))->asBag();
 var temp2 : Set(Property) := cClass.ownedAttribute
   ->select(a | cd.allInstances(Class)
     ->exists(c | c.name = a.type.name))->asSet();
 var temp3 : Bag(Class) := temp2
   ->collect(a | a.type.oclAsType(Class));
 temp += temp3->select(c | temp->excludes(c));
 temp := temp->select(c | res->excludes(c));
 res += temp->select(c | res->excludes(c));
 temp := temp->collect(c | tcAlgorythm(cd, res, c ));
 res += temp->select(c | res->excludes(c));
 res := cutBag(res, Bag{});
 return res;
}

query cutBag(in source : Bag(Class), target : Bag
  (Class)) : Bag(Class)
{
 var res : Bag(Class) := Bag{};
 if(source->size()>0) then
 {
  var temp : Class := source->any(c | true);
  if(target->excludes(temp)) then
  {
   res := cutBag(source->excluding(temp), target
     ->including(temp));
  }
  else
  {
   res := cutBag(source->excluding(temp), target);
  }
  endif;
 }
 else
 {
  return target;
 }
 endif;
 return res;
}

query getTransitiveClosure(in mo : Model) : Bag(Class)
{
 var tc : Set(Class) := Set{};
 var tc1 : Bag(Class) := Bag{};
 var tc2 : Bag(Class) := Bag{};
 var packs := mo.ownedElement;
 var cd : Package := mo.getClassDiagram();
 var classes = mo.getClassDiagramClasses();
 var smartcards : Set(Class) := mo.getCards();
 var message : Class
   := classes![getAppliedStereotype
     ('SecureMDD::Message') <> null];
 tc1
```

```
    := getClassesRec(smardcards, object Set(Class){});
  tc2 := tcAlgorythm(cd, tc, smartcards);
  tc := tc1->union(tc2);
  return tc;
}
{
   return object Operation{
    name := 'decode'+source.name;
    type := getClass(source.getModel(), source.name);
    ownedParameter += object Parameter{
     name := 'in';
     type
       := getDataTypePrimitive(source
          .getModel(), 'byte');
     upper := -1;
     lower := 0;
    };
   };
{
  return object Operation{
   name := 'encode'+source.name;
   type
     := getDataTypePrimitive(source.getModel(), 'byte');
   upper := -1;
   lower := 0;
   ownedParameter += object Parameter{
    name := 'in';
    type := source;
   };
  };
```

2.4 Helper methods to transform a class diagram

This section contains auxiliary methods that are used to transform a (platform-independent) class diagram into the platform-specific class diagram. The methods introduced in this section are invoked from the transformations given in section 2.2 and 2.3.

```
import swt.ModelExtensionLanguageQVTParser;
import cdQueries;
import ddQueries;
import commonQueries;
modeltype UML uses 'http://www.eclipse.org/uml2/3.0.0/UML';
library cdTransformations access library cdQueries,
        ddQueries, commonQueries;

helper Set(Class)::removeDupByName() : Set(Class){
 var newSet : Set(Class) := object Set(Class){};
 self->collect(e|
  if(not newSet->exists(ne| ne.name=e.name)) then{
   newSet+=e;
  }else{
   e.destroy();
  }endif
```

```
 );
 return newSet;
}

helper generateTargetClassException(inout mo: Model)
{
 var e : Class;
 if(targetCard(mo))then{
  e:= object Class{ name := 'ISOException';};
 }endif;
 if(targetServer(mo))then{
  e:= object Class{ name := 'ServiceException';};
 }endif;
 if(targetTerminal(mo))then{
  e:= object Class{ name := 'TerminalException';};
 }endif;
 mo.getClassDiagram().packagedElement+=e;
 return
}

helper removeUserStereoType(inout mo : Model)
{
 var cds := mo.getClassDiagramClasses();
 var userSTClasses : Set(Class) := cds
   ->select(c | c
      .getAppliedStereotype
        ('SecureMDD::Usermessage') <> null);
 log('usermessage: ' + userSTClasses->size().repr());
 userSTClasses
   ->collect(c | destroyUserMessageClass(c));
 return;
}

helper removeAllUnusedManualClasses(inout mo: Model){
 if(targetServer(mo))then{
 var serversUsage : Set(Class)
   := mo.getServers().clientDependency[Usage]
      .target[Class]->asSet();
 var manualClasses : Set(Class)
   :=mo.getClassDiagramClasses()
     ->select(c| c
       .getAppliedStereotype
         ('SecureMDD::Manual')<>null);
 manualClasses:=manualClasses->reject(c| serversUsage
   ->exists(s|s.name=c.name));
 manualClasses->destroy();
 return;} endif;
 if(targetTerminal(mo))then{
 var terminalsUsage : Set(Class)
   := mo.getTerminals().clientDependency[Usage]
      .target[Class]->asSet();
 var manualClasses : Set(Class)
   :=mo.getClassDiagramClasses()
     ->select(c| c
```

```
        .getAppliedStereotype
          ('SecureMDD::Manual')<>null);
 manualClasses:=manualClasses
   ->reject(c| terminalsUsage
     ->exists(s|s.name=c.name));
 manualClasses->destroy();
 return;
 }endif;
 if(targetCard(mo))then{
 var manualClasses : Set(Class)
   :=mo.getClassDiagramClasses()
     ->select(c| c
       .getAppliedStereotype
         ('SecureMDD::Manual')<>null);
 if (manualClasses->isEmpty()) then { return; } endif;
 var smartcardsUsage : Set(Class)
   := mo.getCards().clientDependency[Usage]
     .target[Class]->asSet();
 manualClasses:=manualClasses
   ->reject(c| smartcardsUsage
     ->exists(s|s.name=c.name));
 if (manualClasses->isEmpty()) then { return; } endif;
 manualClasses->destroy();
 return;
 }endif;
}

helper destroyUserMessageClass(inout source : Class)
{
 var cds := source.getModel().getClassDiagramClasses();
 var temp : Bag(Class) := cds
   ->select(c | c.generalization
     ->exists(g | g.general = source))->asBag();
 if(temp->size()>0) then
 {
  temp->collect(c | destroyUserMessageClass(c));
 }
 endif;
 source.destroy();
 return;
}

helper applyTypes(inout mo : Model)
{
 var byte : PrimitiveType
   := object PrimitiveType{name:='byte';};
 mo.getSecurityDatatypes().packagedElement+= byte;
if(targetServer(mo) or targetTerminal(mo))then{
 var p: PrimitiveType
   := getDataTypePrimitive(mo,'Number');
 if( p<>null) then{
  p.name:='int';
 }
 endif;
```

```
 p:= getDataTypePrimitive(mo,'Boolean');
 if( p<>null) then{
  p.name:='boolean';
 }endif;
 mo.getSecurityDatatypes().ownedElement[Class]
   ->applyTypesClass();
 return;}endif;
 if(targetCard(mo))then {
 mo.getClassDiagramClasses()->applyTypesClass();
 mo.getSecurityDatatypes().ownedElement[Class]
   ->applyTypesClass();
 getDataTypePrimitive(mo,'String').destroy();
 var p: PrimitiveType
   := getDataTypePrimitive(mo,'Number');
 if( p<>null) then{
  p.name:='short';
 }
 endif;
 p:= getDataTypePrimitive(mo,'Boolean');
 if( p<>null) then{
  p.name:='boolean';
 }endif;
 return; }endif;
}

helper Class::applyTypesClass() : Class
{
 self.ownedAttribute
   ->collect(a | applyTypesProperty(a));
 self.ownedOperation->collect(o | o.ownedParameter
   ->collect(p | applyTypesParameter(p)));
 return self;
}

helper applyTypesProperty
  (inout source : Property) :Property
{
 if(source.type = getDataTypePrimitive(source
   .getModel(), 'String')) then
 {
  source.type
    := getDataTypePrimitive(source.getModel(), 'byte');
  source.upper := -1;
  source.lower := 0;
 }
 endif;
 return source;
}

helper applyTypesParameter
  (inout source : Parameter) : Parameter
{
 if(source.type = getDataTypePrimitive(source
   .getModel(), 'String')) then
```

```
 {
  source.type
    := getDataTypePrimitive(source.getModel(), 'byte');
  source.upper := -1;
  source.lower := 0;
 }
 endif;
 return source;
}

helper handleLists(inout mo : Model)
{
 var a2List : Set(Property)
   := mo.getClassDiagramClasses()
     ->collect(c | c.ownedAttribute)->asSet();
 a2List->collect(p | check4List(mo, p));
 return
}

helper generateAllInterfaces(inout mo:Model){
 var pack: Set(PackageableElement)
   := mo.getClassDiagram().packagedElement;
 mo.getClassDiagram()
   .packagedElement+= generateInterface4Codeable(mo);
 if(hashUsed(mo) or macUsed(mo)) then
  mo.getClassDiagram()
    .packagedElement += generateInterface4Hash()
 endif;
 if(signUsed(mo) or macUsed(mo)) then {
  mo.getClassDiagram()
    .packagedElement+=generateInterface4Sign();
 }
 endif;
 if(symEncUsed(mo) or asymEncUsed(mo) or macUsed
   (mo)) then
  mo.getClassDiagram()
    .packagedElement+=generateInterface4PlainData()
 endif;
 return;
}

helper check4List
  (inout mo: Model, inout source : Property) : Class
{
 if((source.upper>1 or source.upper=*) and not source
   .type.oclIsTypeOf(PrimitiveType)) then
 {
  var listClass : Class;
  var keys : Set(TypedElement)
    := source.type.ownedElement[TypedElement]
      ->select(e|e.hasStereotype("SecureMDD::key"));
  listClass
    := generateListObject(source.type.name, source
      .type, keys->any(true).type,mo);
```

```
  var listClassAssociation : Association
    := object Association{};
  var targetClass : Class
    := mo.getClassDiagramClasses()
      ->any(c | c.name = source.type.name);
  var sourceEnd : Property := object Property{name
    := 'elems'; visibility
      := VisibilityKind::private; association
        := listClassAssociation; aggregation
          := AggregationKind::none; lower
            := source.lower; upper
              := source.upper; type
                := targetClass; isStatic := true;};
  var listEnd : Property
    := object Property{visibility
      := VisibilityKind::private; association
        := listClassAssociation; aggregation
          := AggregationKind::none; type := listClass};
  listClass.ownedAttribute += sourceEnd;
  listClassAssociation.ownedEnd += listEnd;
  mo.getClassDiagram()
    .packagedElement += listClassAssociation;
  source.type := listClass;
  source.upper := 1;
  source.lower := 1;
  return listClass;
 }
 endif;
 return null;
}

helper generateInterface4Codeable
  (inout mo : Model) : Interface
{
 var iCodeable : Interface := object Interface {
   name := 'Codeable';
   if(targetServer(mo) or targetTerminal(mo)) then{
   ownedOperation += object Operation {name
     := 'getCode'; ownedParameter += object Parameter{name
       := 's'; type
         := getDataTypePrimitive(mo, 'int');};};}
    else {
   ownedOperation += object Operation {name
     := 'getCode'; ownedParameter += object Parameter{name
       := 's'; type
         := getDataTypePrimitive(mo, 'short');};};
     }endif;
   };
 var cd : Package := mo.getClassDiagram();
 if(mo.getAllInterfaces()
   ->exists(i | i.name = 'PlainData')) then
 {
  var iPlainData : Interface
    := getInterface(mo,'PlainData');
```

```
  cd
    .packagedElement += object Usage{client+=
                    iPlainData; supplier+=iCodeable;};
 }
 endif;
 if(mo.getAllInterfaces()
   ->exists(i | i.name = 'Hashed')) then
 {
  var iHashed : Interface := getInterface(mo,'Hashed');
  cd
    .packagedElement += object Usage{client+=
                    iHashed; supplier+=iCodeable;};
 }
 endif;
 var applyData : Set(Class)
   := mo.getClassDiagramClasses()
     ->select(c | c
       .getAppliedStereotype
         ('SecureMDD::data')<>null);
 applyData
   ->collect(c | generateRealizations4Interfaces
     (c, iCodeable));
 var applyMessage : Set(Class)
   := mo.getClassDiagramClasses()
     ->select(c | c
       .getAppliedStereotype
         ('SecureMDD::Message')<>null);
 applyMessage
   ->collect(c | generateRealizations4Interfaces
     (c, iCodeable));
 return iCodeable;
}

helper generateInterface4Hash() : Interface
{
 return object Interface{name:='HashData';};
}

helper generateInterface4Sign() : Interface
{
 return object Interface{name:='SignData';};
}

helper generateInterface4PlainData() : Interface
{
 return object Interface{name:='PlainData';};
}

helper generateListObject
  (source : String, valueType : Type, keyType : Type,
   inout mo : Model) : Class
{
 if(mo.getClassDiagramClasses()
   ->exists(c|c.name='ListOf'+source)) then{
```

```
  return mo.getClassDiagramClasses()
    ->any(c|c.name='ListOf'+source);
 }else{
  var listObject : Class := object Class{name
    := 'ListOf'+source};
  var p1 : Parameter := object Parameter{name
    := 'p'; type := valueType};
  var p2 : Parameter := object Parameter{name
    := 'p'; type := valueType};
  listObject.ownedOperation += object Operation{name
    := 'hasFree'; type
      := getDataTypePrimitive(valueType
        .getModel(), 'boolean')};
  listObject.ownedOperation += object Operation{name
    := 'add'; ownedParameter+=p1;};
  listObject.ownedOperation += object Operation{name
    := 'contains'; ownedParameter+=p2;type
      := getDataTypePrimitive(valueType
        .getModel(), 'boolean');};
  listObject.ownedOperation += object Operation{name
    := 'size'; type
      := getDataTypePrimitive(valueType
        .getModel(), 'int')};
  if(not keyType.oclIsUndefined()) then{
   var key1 : Parameter := object Parameter{name
     := 'key'; type := keyType};
   var key2 : Parameter := object Parameter{name
     := 'key'; type := keyType};
   var value : Parameter := object Parameter{name
     := 'value'; type := valueType};
   listObject.ownedOperation += object Operation{name
     := 'get'; ownedParameter+=key1; type
       := value.type; };
   listObject.ownedOperation += object Operation{name
     := 'set'; ownedParameter+=value;};
   listObject.ownedOperation += object Operation{name
     := 'containsKey'; ownedParameter+=key2; type
       := getDataTypePrimitive(valueType
         .getModel(), 'boolean');};
   listObject.ownedOperation += object Operation{name
     := 'remove'; ownedParameter+=key1; type
       := getDataTypePrimitive(valueType
         .getModel(), 'boolean')};
  }endif;
  mo.getClassDiagram().packagedElement += listObject;
  return listObject;
 }endif;
 return null;
}

helper generateRealizations4Interfaces
  (inout source : Class, interface : Interface)
{
 var iRealization : InterfaceRealization
```

```
    := object InterfaceRealization{contract
      := interface; implementingClassifier := source;};
 source.interfaceRealization += iRealization;
 return;
}

helper generateCommonMissingClassesPart1
  (inout mo:Model){
 logging('generateClassCoding');
 generateClassCoding(mo);
 logging('generateClassCode');
 generateClassCode(mo);
 logging('generateClassArrays');
 generateClassArrays(mo);
 logging('generateClassState');
 generateClassState(mo);
 logging('generateClassGate');
 generateClassGate(mo);
 if(targetServer(mo) or targetTerminal(mo))then{
  logging("generateClassLengthConstants");
  generateClassLengthConstants(mo);
 }endif;
}

helper generateCommonMissingClassesPart2
  (inout mo:Model){
 generateMessageWrapper(mo);
 generateClassPorts(mo);
 generateTargetClassException(mo);
}

helper generateClassGate(inout mo:Model)
{
 var c: Class:=object Class
 {
   name:='Gate';
   visibility:=VisibilityKind::public;
 };
 mo.getClassDiagram().packagedElement+=c;
 return;
}

helper generateClassState(inout mo:Model)
{
 var c: Class:=object Class
 {
   name:='State';
   visibility:=VisibilityKind::public;
 };
 mo.getClassDiagram().packagedElement+=c;
 return;
}

helper generateClassArrays(inout mo:Model){
```

```
var c:Class:=object Class
 {
  name:='Arrays';
  visibility:=VisibilityKind::public;
 };
 mo.getClassDiagram().packagedElement+=c;
 insertDefaultArraysOperation(mo);
 return;
}

helper handleAllInterfaces(inout mo :Model){
    handleHash(mo);
     handleSign(mo);
     handlePlainData(mo);
     handleCodeable(mo);
}

helper handleHash(inout mo : Model)
{
 var applyH : Set(Class)
   := mo.getClassDiagramClasses()
     ->select(c | c
       .getAppliedStereotype
         ('SecureMDD::HashData')<>null);
 if(applyH->size()>0) then
  applyH
    ->collect(c | generateRealizations4Interfaces
      (c, getInterface(mo,'HashData')))
 endif;
 return
}

helper handleSign(inout mo : Model)
{
 var applyS : Set(Class)
   := mo.getClassDiagramClasses()
     ->select(c | c
       .getAppliedStereotype
         ('SecureMDD::SignData')<>null);
 if(applyS->size()>0) then
  applyS
    ->collect(c | generateRealizations4Interfaces
      (c, getInterface(mo,'SignData')))
 endif;
 return
}

helper handlePlainData(inout mo : Model)
{
 var applyPD : Set(Class)
   := mo.getClassDiagramClasses()
     ->select(c | c
       .getAppliedStereotype
         ('SecureMDD::PlainData')<>null);
```

```
 if(hashUsed(mo) and macUsed
   (mo)) then applyPD += getSecureDataType
     (mo, 'HashedData') endif;
 if(applyPD->size()>0) then
  applyPD
    ->collect(c | generateRealizations4Interfaces
      (c, getInterface(mo,'PlainData')))
 endif;
 return
}

helper handleCodeable(inout mo : Model)
{
 if(targetServer(mo))then{
  mo.getCodeableAndCompareableClasses()
    ->collect(c | generateRealizations4Interfaces
      (c, getInterface(mo, 'Codeable')));
 }endif;
 if(targetCard(mo) or targetTerminal(mo))then{
  mo.getClassDiagramClasses()
    ->select(e|e.implementsCodeable(mo))
      ->collect(c | generateRealizations4Interfaces
        (c, getInterface(mo, 'Codeable')));
 }endif;
 return
}

helper Model::getCodeableAndCompareableClasses
  () :OrderedSet(Class){
 var tc : OrderedSet(Class)
   := self.getClassDiagramClasses()
     ->select(e|e.isCodeable())
       ->union(self.getCards()
         ->reject(e|e.isAbstract))->asOrderedSet()
           ->sortedBy(name);
 var comp : Set(Class)
   := getReachableClasses(self.getServers());
 var comp1: Set(Class):=comp->reject(e|tc
   ->exists(t|e.name=t.name) or e
     .isSimpleCommClass() or e.isServer() or e.name
       .startsWith("ListOf") or e
         .hasStereotype("SecureMDD::Manual"));
 tc+=comp1->asOrderedSet()->sortedBy(name);
 return tc;
}

helper generateHashedDataClass(inout mo : Model)
{
 var cd : Package := mo.getClassDiagram();
 var cds := mo.getClassDiagramClasses();
 if(hashUsed(mo)
  or macUsed(mo)) then
 {
  var codingClass : Class := cds![name='Coding'];
```

```
  var hdClass : Class
    := getSecureDataType(mo, 'HashedData');
  var iCodeable : Interface
    := cd.ownedElement[Interface]
      ->any(i | i.name='Codeable');
  hdClass.ownedAttribute
    ->any(e | e.name.match('hashed')).destroy();
  hdClass.ownedAttribute += object Property{name
    :='hashed'; type
      :=getDataTypePrimitive(mo, 'byte'); upper
        := -1; lower := 0; visibility
          := VisibilityKind::private;};
  hdClass.ownedOperation += object Operation{name
    :='copy'; ownedParameter+=object Parameter{name
      :='from';type:=hdClass;};};
  hdClass.ownedOperation += object Operation{name
    :='equals'; ownedParameter+=object Parameter{name
      :='other';type:=iCodeable;};};
  hdClass.ownedOperation += object Operation{name
    :='hash';
   ownedParameter+=object Parameter{name:='h';type
     :=getInterface(mo, 'HashData');};
   type:=hdClass;};
  generateRealizations4Interfaces(hdClass, iCodeable
    .oclAsType(Interface));
  applyAssociations(mo, hdClass, 'SecureMDD::hashed');
  var attrAssociation : Association
    := object Association{};
  var sourceProperty : Property
    := object Property{name:='c'; visibility
      := VisibilityKind::private; association
        := attrAssociation; aggregation
          :=AggregationKind::none; type
            := codingClass; lower := 1; upper := 1;};
  var aEnd : Property := object Property{visibility
    := VisibilityKind::private; association
      := attrAssociation; aggregation
        := AggregationKind::none; type := hdClass};
  attrAssociation.ownedEnd += aEnd;
  hdClass.ownedAttribute += sourceProperty;
  cd.packagedElement += attrAssociation;
 }
 endif;
 return
}

helper generateSignedDataClass(inout mo : Model)
{
 var cd : Package := mo.getClassDiagram();
 var cds := mo.getClassDiagramClasses();
 if(signUsed(mo)) then
 {
  var codingClass : Class := cds![name='Coding'];
  var sdClass : Class
```

```
    := getSecureDataType(mo, 'SignedData');
  var iCodeable : Interface
    := cd.ownedElement[Interface]
      ->any(i | i.name='Codeable');
  sdClass.ownedAttribute
    ->any(e | e.name.match('signed')).destroy();
  sdClass.ownedAttribute += object Property{name
    :='signed'; type
      :=getDataTypePrimitive(mo, 'byte'); upper
        := -1; lower := 0; visibility
          := VisibilityKind::private;};
  sdClass.ownedOperation += object Operation{name
    :='copy'; ownedParameter+=object Parameter{name
      :='from';type:=sdClass;};};
  sdClass.ownedOperation += object Operation{name
    :='equals'; ownedParameter+=object Parameter{name
      :='other';type:=iCodeable;};};
  sdClass.ownedOperation += object Operation{name
    :='sign';
   ownedParameter+=object Parameter{name:='key';type
     :=getSecureDataType(mo, 'Key');};
   ownedParameter+=object Parameter{name:='d';type
     :=getInterface(mo, 'SignData');};
   type:=sdClass;};
  sdClass.ownedOperation += object Operation{name
    :='verify';
   ownedParameter+=object Parameter{name:='key';type
     :=getSecureDataType(mo, 'Key');};
   ownedParameter+=object Parameter{name:='sd';type
     :=sdClass;};
   ownedParameter+=object Parameter{name:='s';type
     :=getInterface(mo, 'SignData');};
   type:=getDataTypePrimitive(mo, 'boolean');};
  generateRealizations4Interfaces(sdClass, iCodeable
    .oclAsType(Interface));
  applyAssociations(mo, sdClass, 'SecureMDD::signed');
 }
 endif;
 return
}

helper generateEncClasses(inout mo : Model)
{
 if(symEncUsed(mo) or asymEncUsed(mo)
  or macUsed(mo)) then
 {
  var cd : Package := mo.getClassDiagram();
  var cds := cd.ownedElement[Class];
  var is := cd.ownedElement[Interface];
  var codingClass : Class := cds
    ->any(c | c.name='Coding');
  var encClass : Class
    := getSecureDataType(mo, 'EncData');
  var iCodeable : Interface := is
```

```
  ->any(i | i.name='Codeable');
var iPlain : Interface := is
  ->any(i | i.name='PlainData');
encClass.ownedAttribute
  ->any(e | e.name.match('encrypted')).destroy();
encClass.ownedAttribute += object Property{name
  :='encrypted'; type
    :=getDataTypePrimitive(mo, 'byte'); upper
      := -1; lower := 0; visibility
        := VisibilityKind::private;};
encClass.ownedOperation += object Operation{name
  :='copy'; ownedParameter+=object Parameter{name
    :='o';type:=encClass;};};
encClass.ownedOperation += object Operation{name
  :='equals'; ownedParameter+=object Parameter{name
    :='other';type:=iCodeable;};};
encClass.ownedOperation += object Operation{name
  :='isEncDataSymm'; type
    :=getDataTypePrimitive(mo, 'boolean');};
encClass.ownedOperation += object Operation{name
  :='isEncDataAsymm'; type
    :=getDataTypePrimitive(mo, 'boolean');};
generateRealizations4Interfaces(encClass, iCodeable
  .oclAsType(Interface));
if(symEncUsed(mo)
 or macUsed(mo)) then
{
 var encSymmClass : Class
   := getSecureDataType(mo, 'EncDataSymm');
 encSymmClass.ownedOperation += object Operation{
  name:='encrypt';
  type:=encSymmClass;
  ownedParameter+=object Parameter{name:='k'; type
    :=getSecureDataType(mo, 'SymmKey');};
  ownedParameter+=object Parameter{name
    :='plain'; type:=iPlain;};
  };
 encSymmClass.ownedOperation += object Operation{
  name:='decrypt';
  type:=iPlain;
  ownedParameter+=object Parameter{name:='k'; type
    :=getSecureDataType(mo, 'SymmKey');};
  ownedParameter+=object Parameter{name:='e'; type
    :=encSymmClass;};
  };
 applyAssociations
   (mo, encSymmClass, 'SecureMDD::encrypted');
 var attrAssociation : Association
   := object Association{};
 var sourceProperty : Property
   := object Property{name:='c'; visibility
     := VisibilityKind::private; association
       := attrAssociation; aggregation
         :=AggregationKind::none; type
```

```
            := codingClass; lower := 1; upper := 1;};
  var aEnd : Property := object Property{visibility
    := VisibilityKind::private; association
      := attrAssociation; aggregation
        := AggregationKind::none; type
          := encSymmClass};
  attrAssociation.ownedEnd += aEnd;
  encSymmClass.ownedAttribute += sourceProperty;
  cd.packagedElement += attrAssociation;
 }
 endif;
 if(asymEncUsed(mo)) then
 {
  var encAsymmClass : Class
    := getSecureDataType(mo, 'EncDataAsymm');
  encAsymmClass.ownedOperation += object Operation{
   name:='encrypt';
   type:=encAsymmClass;
   ownedParameter+=object Parameter{name:='k'; type
     :=getSecureDataType(mo, 'Publickey');};
   ownedParameter+=object Parameter{name
     :='plain'; type:=iPlain;};
   };
  encAsymmClass.ownedOperation += object Operation{
   name:='decrypt';
   type:=iPlain;
   ownedParameter+=object Parameter{name:='k'; type
     :=getSecureDataType(mo, 'Privatekey');};
   ownedParameter+=object Parameter{name:='e'; type
     :=encAsymmClass;};
   };
  applyAssociations
    (mo, encAsymmClass, 'SecureMDD::encryptedAsymm');
  var attrAssociation : Association
    := object Association{};
  var sourceProperty : Property
    := object Property{name:='c'; visibility
      := VisibilityKind::private; association
        := attrAssociation; aggregation
          :=AggregationKind::none; type
            := codingClass; lower := 1; upper := 1;};
  var aEnd : Property := object Property{visibility
    := VisibilityKind::private; association
      := attrAssociation; aggregation
        := AggregationKind::none; type
          := encAsymmClass};
  attrAssociation.ownedEnd += aEnd;
  encAsymmClass.ownedAttribute += sourceProperty;
  cd.packagedElement += attrAssociation;
 }
 endif;
}
endif;
return;
```

```
}

helper generateMACClass(inout mo : Model)
{
 if(macUsed(mo)) then
 {
  var cd : Package := mo.getClassDiagram();
  var cds := cd.ownedElement[Class];
  var is := cd.ownedElement[Interface];
  var sd : Package := mo.getSecurityDatatypes();
  var sds := sd.ownedElement[Class];
  var sis := sd.ownedElement[Interface];
  var macClass : Class
    := getSecureDataType(mo, 'MACData');
  generateRealizations4Interfaces
    (macClass, getInterface(mo, 'Codeable'));
  if(symEncUsed(mo) or asymEncUsed(mo)) then
   generateRealizations4Interfaces
     (macClass, getInterface(mo, 'PlainData'))
  endif;
  if(hashUsed(mo)) then
   generateRealizations4Interfaces
     (macClass, getInterface(mo, 'HashData'))
  endif;
  macClass.ownedOperation += object Operation{
   name:='computeMAC';
   type:=macClass;
   ownedParameter+=object Parameter{name
     :='encKey'; type
       :=getSecureDataType(mo, 'SymmKey');};
   ownedParameter+=object Parameter{name
     :='macKey'; type
       :=getSecureDataType(mo, 'SymmKey');};
   ownedParameter+=object Parameter{name:='pd'; type
     :=getInterface(mo, 'PlainData');};
   };
  macClass.ownedOperation += object Operation{
   name:='decryptMAC';
   type:=getInterface(mo, 'PlainData');
   ownedParameter+=object Parameter{name
     :='encKey'; type
       :=getSecureDataType(mo, 'SymmKey');};
   ownedParameter+=object Parameter{name
     :='macKey'; type
       :=getSecureDataType(mo, 'SymmKey');};
   ownedParameter+=object Parameter{name:='md'; type
     :=macClass;};
   };
  macClass.ownedOperation += object Operation{name
    :='copy'; ownedParameter+=object Parameter{name
      :='o';type:=macClass;};};
  macClass.ownedOperation += object Operation{name
    :='equals'; ownedParameter+=object Parameter{name
      :='other';type:=getInterface(mo, 'Codeable');};};};
```

```
  macClass.ownedOperation += object Operation{name
    :='isMACData'; type
      :=getDataTypePrimitive(mo, 'boolean');};
  applyAssociations(mo, macClass, 'SecureMDD::MAC');
 }
 endif;
 return;
}

helper generateConstructorForTarget
  (inout source: Class) {
 var ops : OrderedSet(Operation)
   := object OrderedSet(Operation){};
 ops += object Operation{name:= source.name};
 var attrs : OrderedSet(Property)
   := source.getAllAttributesForInitialization()
     ->asOrderedSet()->sortedBy(name);
 if(not attrs->isEmpty()) then {
  var res : Operation := object Operation{name
    := source.name};
  attrs->collect(att|
   if(true) then{
    res.ownedParameter +=
    object Parameter{name:=att.name; type
      :=att.type.translateType();};
   }endif
  );
  ops += res;
 } endif;
 source.ownedOperation+=ops;
 return;
}

helper buildConstructors(inout mo : Model)
{
 var cs : Set(Class) := mo.getClassDiagramClasses();
 var cs1 : Set(Class) := cs->select(c|
   not c.hasStereotype('SecureMDD::Constant') and
   not mo.getTargetClasses()->includes(c)
 );
 if(targetCard(mo)) then{
  cs1:=cs1
    ->select(c|not (c.name = 'Store') and not (c
      .name = 'Math'));
 }endif;
 cs1->collect(c | generateConstructor(c));
  mo.getTargetClasses()
    ->collect(c | generateConstructorForTarget(c));
 return;
}

helper generateConstructor(inout source: Class) : Class
{
 var cds := source.getModel().getClassDiagramClasses();
```

```
 var sourceAttributesWithClassName : Set(Property)
   := source.ownedAttribute[cds
     ->exists(c | c.name = type.name)];
 var sourceClassAttributes : Bag(Class)
   := sourceAttributesWithClassName
     ->collect(a | a.type.oclAsType(Class));
 var ops : OrderedSet(Operation)
   := object OrderedSet(Operation){};
 ops += object Operation{name:= source.name};
 if( sourceClassAttributes
   ->size()>1 or (source.ownedOperation
     ->exists(o|o.name=source.name and o
       .ownedParameter->notEmpty()))) then
 {
 }
 else
 {
  if(sourceClassAttributes->size()=1) then
  {
   var res : Operation := object Operation{name
     := source.name};
   res.ownedParameter += (source.ownedAttribute
     ->select(a | sourceAttributesWithClassName
       ->excludes(a)))
         ->collect(p | generateParameter4Constructor
           (p));
   res
     .ownedParameter += generateParameter4Constructor
       (sourceAttributesWithClassName->any(a | true));
   ops += res;
  }
  else
   if(source.ownedAttribute->size()>0) then
   {
    var res : Operation := object Operation{name
      := source.name};
    res.ownedParameter += source.ownedAttribute
      ->collect(p | generateParameter4Constructor(p));
    ops += res;
   }
   endif
  endif;
 }
 endif;
 source.ownedOperation+=ops;
 return source;
}

helper generateParameter4Constructor
  (inout source : Property) : Parameter
{
 if(source.upper = 1 and source.lower = 1) then
 {
  return object Parameter{name:= source.name; type
```

```
    := source.type;};
 }
 endif;
 return object Parameter{name:= source.name; type
   := source.type; upper:=1; lower:= source.lower;};
}

helper applyAssociations
  (inout mo : Model, encClass : Class, st : String)
{
 var allEncAt : Set(Class)
   := mo.getClassDiagramClasses()
     ->select(c | c.ownedAttribute
       ->exists(a | a
         .getAppliedStereotype(st) <> null));
 //var allEncAs : Bag(Property)
   :=mo.getClassDiagram().ownedElement[Property]
     .oclAsType(Property);
 allEncAt
   ->collect(c | changeClassAttr(c, encClass, st));
 return;
}

helper changeClassAttr
  (inout source : Class, inout encClass : Class, st : String)
{
 var mo :Model := source.getModel();
 var cd : Package := mo.getClassDiagram();
 var oldProperty : Property := getEncAttr(source, st);
 if (oldProperty = null ) then return endif;
 var attrAssociation : Association
   := object Association{};
 var oldPropertyType : Class
   := mo.getClassDiagramClasses()
     ->any(c | oldProperty.type.name = c.name);
 cd
   .packagedElement += object Usage{supplier += oldPropertyType;
                                    client += source};
 var sourceProperty : Property
   := object Property{name
     :=oldProperty.name; visibility
       := oldProperty.visibility; association
         := attrAssociation; aggregation
           :=oldProperty.aggregation; lower
             := oldProperty.lower; upper
               := oldProperty.upper; type
                 := encClass; isStatic
                   := oldProperty.isStatic;};
 var aEnd : Property := object Property{visibility
   := VisibilityKind::private; association
     := attrAssociation; aggregation
       := AggregationKind::none; type := source};
 attrAssociation.ownedEnd += aEnd;
 source.ownedAttribute += sourceProperty;
```

```
 oldProperty.getAppliedStereotypes()
   ->collect(s|if(s.qualifiedName <> st) then {
  sourceProperty.applyStereotype(s);
 }endif);
 oldProperty.association.destroy();
 oldProperty.destroy();
 cd.packagedElement += attrAssociation;
 changeClassAttr( source, encClass, st );
}

helper generateClassCoding(inout mo : Model) {
  var c: Class:= object Class {name := 'Coding';};
  mo.getClassDiagram().packagedElement += c;
     insertDefaultCodingClassOperations(mo);
 return;
}

helper generateClassSWTReader(inout mo: Model)
{
 var swtreader : Class:= object Class{ name
   := 'SWTReader';};
 mo.getClassDiagram().packagedElement+=swtreader;
 return
}

helper generateClassCode(inout mo: Model)
{
 var code : Class:= object Class{ name := 'Code';};
 mo.getClassDiagram().packagedElement+=code;
 if(targetServer(mo) or targetTerminal(mo))then{
 var numberType : PrimitiveType
   := getDataTypePrimitive(mo, 'int'); }endif;
 if(targetCard(mo))then{
 var numberType : PrimitiveType
   := getDataTypePrimitive(mo, 'short');
 }endif;
 var constants : Class := mo.getClassDiagramClasses()
   ->any(c | c
     .getAppliedStereotype
       ('SecureMDD::Constant') <> null)
         .oclAsType(Class);
 var prop : OrderedSet(Property);
 prop
   := generateDefaultShortValues4Constants(constants
     .ownedAttribute, 1);
 constants.ownedAttribute.destroy();
 constants.ownedAttribute += prop;
 var byteType : Type
   := getDataTypePrimitive(mo, 'byte');
 var c : Integer := 0;
 code.ownedAttribute += object Property{
  name:='IGNORE';
  _default:=(c).toString();
  type:=byteType;
```

```
 visibility:=VisibilityKind::public;
 isLeaf:=true;
 isStatic:=true;
}; c:=c+1;
code.ownedAttribute += object Property{
 name:='NULL';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
}; c:=c+1;
 code.ownedAttribute += object Property{
 name:='BOOLEAN';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
}; c:=c+1;
 code.ownedAttribute += object Property{
 name:='BYTE';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
}; c:=c+1;
 code.ownedAttribute += object Property{
 name:='INT';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
}; c:=c+1;
 code.ownedAttribute += object Property{
 name:='BOOLEANARRAY';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
}; c:=c+1;
 code.ownedAttribute += object Property{
 name:='BYTEARRAY';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
}; c:=c+1;
code.ownedAttribute += object Property{
 name:='SHORTARRAY';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
}; c:=c+1;
code.ownedAttribute += object Property{
```

```
 name:='NONCE';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
}; c:=c+1;
code.ownedAttribute += object Property{
 name:='SECRET';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
}; c:=c+1;
code.ownedAttribute += object Property{
 name:='SYMMKEY';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
}; c:=c+1;
code.ownedAttribute += object Property{
 name:='PRIVATEKEY';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
}; c:=c+1;
code.ownedAttribute += object Property{
 name:='PUBLICKEY';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
}; c:=c+1;
code.ownedAttribute += object Property{
 name:='HASHEDDATA';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
}; c:=c+1;
code.ownedAttribute += object Property{
 name:='SIGNEDDATA';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
}; c:=c+1;
code.ownedAttribute += object Property{
 name:='ENCDATASYMM';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
```

```
}; c:=c+1;
code.ownedAttribute += object Property{
 name:='ENCDATAASYMM';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
}; c:=c+1;
code.ownedAttribute += object Property{
 name:='MACDATA';
 _default:=(c).toString();
 type:=byteType; visibility
   :=VisibilityKind::public; isLeaf:=true; isStatic
     :=true;
}; c:=c+1;
if(targetServer(mo))then{
var tc : OrderedSet(Class)
  := mo.getCodeableAndCompareableClasses();
var codeAttr : OrderedSet(Property)
  := generateDefaultByteValues4Code(tc, c)
    ->asOrderedSet();
code.ownedAttribute += codeAttr; }endif;
if(targetTerminal(mo))then{
var usermessages:OrderedSet(Class)
  := mo.getClassDiagramClasses()
    ->select(e|e
      .hasStereotype("SecureMDD::Usermessage")
 or e.general
   ->select(g|g
     .hasStereotype("SecureMDD::Usermessage"))
       ->notEmpty())->sortedBy(name);
var tcWithoutUsermessages : OrderedSet(Class)
  := mo.getClassDiagramClasses()->select(e|
 e.isCodeable() and not usermessages
   ->exists(u|u.name=e.name) )->union(mo.getCards()
     ->reject(e|e.isAbstract))->asOrderedSet()
       ->sortedBy(name);
var codeAttr1 : OrderedSet(Property)
  := generateDefaultByteValues4Code
    (tcWithoutUsermessages, c)->asOrderedSet()
      ->sortedBy(name);
var codeAttr2 : OrderedSet(Property)
  := generateDefaultByteValues4Code
    (usermessages, c+tcWithoutUsermessages->size())
      ->asOrderedSet()->sortedBy(name);
code.ownedAttribute += codeAttr1;
code.ownedAttribute += codeAttr2; }endif;
if(targetCard(mo))then{
var tc : OrderedSet(Class)
  := mo.getClassDiagramClasses()
    ->select(e|e.isCodeable())->union(mo.getCards()
      ->reject(e|e.isAbstract))->asOrderedSet()
        ->sortedBy(name);
var codeAttr : OrderedSet(Property)
```

```
    := generateDefaultByteValues4Code(tc, c)
      ->asOrderedSet()->sortedBy(name);
 code.ownedAttribute += codeAttr;}endif;
 return;
}

helper generateDefaultShortValues4Constants
  (inout source: OrderedSet
    (Property), dValue : Integer) : OrderedSet
      (Property)
{
 var params:OrderedSet(Property);
 var value:Integer:=dValue;
 source->collect(p|
  if(true) then{
   params+=object Property{
    name:=p.name;
    visibility:=VisibilityKind::public;
    isStatic:=true;
    isLeaf:=true;
    type :=getDataTypePrimitive(p.getModel(), 'short');
    defaultValue:=object OpaqueExpression{_body
      :=value.toString()};
   };
   value:=value+1;
  }endif
 );
 return params;
}

helper generateDefaultByteValues4Code
  (inout source: OrderedSet
    (Class), dValue : Integer) : OrderedSet(Property)
{
 var params:OrderedSet(Property);
 var value:Integer:=dValue;
 source->collect(p|
  if(true) then{
   params+=object Property{
    name:=p.name.toUpper();
    visibility:=VisibilityKind::public;
    isStatic:=true;
    isLeaf:=true;
    type :=getDataTypePrimitive(p.getModel(), 'byte');
    defaultValue:=object OpaqueExpression{_body
      :=value.toString()};
   };
   value:=value+1;
  }endif
 );
 return params;
}

helper Class::implementsCodeable
```

```
  (inout mo:Model) : Boolean {
 if( self
   .getAppliedStereotype
     ('SecureMDD::Message') <> null ) then return true endif;
 if( self.isSimpleCommClass() ) then return true endif;
 if( self.hasStereotype('SecureMDD::HashData') or
  self.hasStereotype('SecureMDD::SignData') or
  self.hasStereotype('SecureMDD::MacData')  or
  self
    .hasStereotype
      ('SecureMDD::PlainData')) then return true endif;
 if( self.implementsInterface('HashData') or
  self.implementsInterface('SignData') or
  self.implementsInterface('MacData') or
  self
    .implementsInterface
      ('PlainData')) then return true endif;
    if( self
      .isNeededCDClassForInitialization
        ()) then return true endif;
          if( self
            .isNeededCDClassForInitialization
              ()) then return true endif;}endif;
 if(targetCard(mo) or targetTerminal(mo))then{
  if( self
    .isCDClassForInitialization
      ()) then return true endif;}endif;
 return self.getModel().getClassDiagramClasses()
   .ownedAttribute
     ->select(e|not (e = self) and e.type = self)
       ->exists(e|e._class.isCodeable());
}

helper generateClassLengthConstants(inout mo: Model)
{
 var lc : Class:= object Class{ name
   := 'LengthConstants';};
 mo.getClassDiagram().packagedElement+=lc;
 var numberType : PrimitiveType
   :=  getDataTypePrimitive(mo, 'int');
 lc.ownedAttribute += object Property{
  name:='LENGTHOFSECRET';
  visibility := VisibilityKind::public;
  _default := getLengthOfSecret().toString();
  type := numberType;
  isLeaf := true;
  isStatic := true;};
 lc.ownedAttribute += object Property{
  name:='LENGTHOFNONCE';
  visibility := VisibilityKind::public;
  _default := getLengthOfNonce().toString();
  type := numberType;
  isLeaf := true;
  isStatic := true;};
```

```
lc.ownedAttribute += object Property{
 name:='LENGTHOFSTRING';
 visibility := VisibilityKind::public;
 _default := getLengthOfString().toString();
 type := numberType;
 isLeaf := true;
 isStatic := true;};
lc.ownedAttribute += object Property{
 name:='MAX_ENCODING_LENGTH_MESSAGES';
 visibility := VisibilityKind::public;
 _default
   := computeMaxEncodingLengthMessages().toString();
 type := numberType;
 isLeaf := true;
 isStatic := true;};
if( asymEncUsed(mo)) then
lc.ownedAttribute+=object Property{
 name:='MAXENCRYPTLENGTHASYMM';
 type:=numberType;
 visibility:=VisibilityKind::public;
 isLeaf:=true;
 isStatic:=true;
 _default:=computeEncDataLength_Asymm().toString();
}
endif;
if( signUsed(mo)) then{
lc.ownedAttribute+=object Property{
 name:='MAXSIGNEDLENGTH';
 type:=numberType;
 visibility:=VisibilityKind::public;
 isLeaf:=true;
 isStatic:=true;
 _default:=computeSignedDataLength().toString();
};
lc.ownedAttribute+=object Property{
 name:='MAX_ENCODING_LENGTH_SIGNDATA';
 type:=numberType;
 visibility:=VisibilityKind::public;
 isLeaf:=true;
 isStatic:=true;
 _default
   :=computeMaxEncodingLengthSignData().toString();
}
}endif;
if(hashUsed(mo) or macUsed(mo)) then {
lc.ownedAttribute += object Property{
 name:='COMPUTEDHASHDATALENGTH';
 visibility := VisibilityKind::public;
 _default := computeHashDataLength().toString();
 type := numberType;
 isLeaf := true;
 isStatic := true;};
} endif;
if (symEncUsed(mo) or macUsed(mo)) then
```

```
 lc.ownedAttribute+=object Property{
  name:='MAXENCRYPTLENGTHSYMM';
  type:=numberType;
  visibility:=VisibilityKind::public;
  isLeaf:=true;
  isStatic:=true;
  _default:=computeEncDataLength_Symm().toString();
 } endif;
 return;
}

helper insertDefaultCodingClassOperations
  (inout mo : Model)
{
 var source : Class := mo.getClassDiagramClasses()
   ->any(c | c.name = 'Coding');
 source.ownedOperation += object Operation{
  name := 'getInstance';
  type := source;
 };
 source.ownedOperation += object Operation{
  name := 'getEncodingLength';
  if(targetServer(mo) or targetTerminal(mo))then
  type := getDataTypePrimitive(mo,'int')endif;
  if(targetCard(mo))then type
    := getDataTypePrimitive(mo,'short')endif;
 };
 source.ownedOperation += object Operation{
  name := 'encode';
 if(targetServer(mo) or targetTerminal(mo))then
  type := getDataTypePrimitive(mo,'int')endif;
 if(targetCard(mo))then type
   := getDataTypePrimitive(mo,'short')endif;
  ownedParameter += object Parameter{
   name := 'c';
   type
     := mo.getClassDiagram().ownedElement[Interface]
       ->any(i | i.name = 'Codeable');
   };
  ownedParameter += object Parameter{
   name := 'destination';
   type := getDataTypePrimitive(mo,'byte');
   upper := -1;
   lower := 0;
   };
 };
 source.ownedOperation += object Operation{
 name := 'decodeMessage';
 type := mo.getClassDiagramClasses()
   ->any(i | i.name = 'Message');
 ownedParameter += object Parameter{
  name := 'in';
  type := getDataTypePrimitive(mo, 'byte');
  upper := -1;
```

```
  lower := 0;
  };
 ownedParameter += object Parameter{
  name := 'offset';
 if(targetServer(mo) or targetTerminal(mo))then
  type := getDataTypePrimitive(mo,'int')endif;
 if(targetCard(mo))then type
   := getDataTypePrimitive(mo,'short')endif;
  };
 ownedParameter += object Parameter{
  name := 'expectedLength';
 if(targetServer(mo) or targetTerminal(mo))then
  type := getDataTypePrimitive(mo,'int')endif;
 if(targetCard(mo))then type
   := getDataTypePrimitive(mo,'short')endif;
  };
 };
 if(symEncUsed(mo) or asymEncUsed(mo)) then
  source.ownedOperation += object Operation{
  name := 'decodePlainData';
  type
    := mo.getClassDiagram().ownedElement[Interface]
      ->any(i | i.name = 'PlainData');
  ownedParameter += object Parameter{
   name := 'in';
   type := getDataTypePrimitive(mo, 'byte');
   upper := -1;
   lower := 0;
   };
  ownedParameter += object Parameter{
   name := 'expectedType';
   type := getDataTypePrimitive(mo, 'byte');
   };
  }
 endif;
 return;
}

helper generateMessageWrapper(inout mo : Model){
 if(not mo.getServers().oclIsInvalid()
   ->forAll(e| e=true)) then{
  var messageWrapper: Class:= object Class {name
    := "MessageWrapper";};
  mo.getClassDiagram()
    .packagedElement += messageWrapper;
  var message:Class:=mo.getClassDiagramClasses()
    ->select(c|c.hasStereotype("SecureMDD::Message"))
      ->any(true);
  var ass: Association:= object Association{
   var client: Property:=object Property{type
     :=messageWrapper; visibility
       :=VisibilityKind::private; };
   var supplier: Property:=object Property{name
     := "msg"; type:=message;visibility
```

```
        :=VisibilityKind::private;};
    ownedEnd+=client;
    ownedEnd+=supplier;
   };
   mo.getClassDiagram().packagedElement+=ass;
   messageWrapper.ownedAttribute+=ass.memberEnd
     ->any(e|e.type.name=message.name);
 }endif;
return;
}

helper generateClassPorts(inout mo: Model)
{
 var portsc : Class:= object Class{ name := 'Ports';};
 mo.getClassDiagram().packagedElement+=portsc;
 var numberType : PrimitiveType
   :=  getDataTypePrimitive(mo, 'int');
 var num:Integer:=0;
 mo.getAllPorts()
   ->collect(p|if(p.name.oclIsUndefined() or p.name
     .match('')) then {
 p.name := p.owner.oclAsType(Node).name.toFirstUpper()
   + '2'
   + p.type.name.toFirstUpper()
   + '_default'
 var allPorts: Set(Property):=mo.getAllPorts();
 num:=0;
 var terminals: Set(Node)
   :=mo.getDeploymentDiagram().ownedElement[Node]
     ->select(n| n
       .hasStereotype("SecureMDD::Terminal"));
 terminals->collect(t| if(true) then{
  var tPorts: OrderedSet(Property)
    := t.ownedAttribute->sortedBy(name);
  var tcPorts : OrderedSet(Property) := tPorts
    ->select(m|m.type
      .hasStereotype("SecureMDD::Smartcard"))
       ->sortedBy(name);
  tcPorts->collect(p| if(true) then {
   portsc.ownedAttribute += object Property{
    name:=p.name;
    visibility := VisibilityKind::public;
    _default := num.toString();
    type := numberType;
    isLeaf := true;
    isStatic := true;};
    num := num+1;
   } endif
  );
  var terminalWithoutCardPorts:OrderedSet(Property)
    := tPorts
      ->select(m| not m.type
        .hasStereotype("SecureMDD::Smartcard"))
         ->sortedBy(name);
```

```
 terminalWithoutCardPorts->collect(p| if(true) then {
 portsc.ownedAttribute += object Property{
  name:=p.name;
  visibility := VisibilityKind::public;
  _default := num.toString();
  type := numberType;
  isLeaf := true;
  isStatic := true;};
  num := num+1;
 } endif
 );
num:=0;
}endif
);
var cards: Set(Node)
  :=mo.getDeploymentDiagram().ownedElement[Node]
    ->select(n| n
      .hasStereotype("SecureMDD::Smartcard"));
cards->collect(c| if(true) then{
 var cPorts: OrderedSet(Property)
   := c.ownedAttribute->sortedBy(name);
 cPorts->collect(p| if(true) then {
  portsc.ownedAttribute += object Property{
   name:=p.name;
   visibility := VisibilityKind::public;
   _default := num.toString();
   type := numberType;
   isLeaf := true;
   isStatic := true;};
   num := num+1;
  } endif
 );
num:=0;
}endif
);
var server: Set(Node)
  :=mo.getDeploymentDiagram().ownedElement[Node]
    ->select(n| n
      .hasStereotype("SecureMDD::Service"));
server->collect(s| if(true) then{
 var sPorts: OrderedSet(Property)
   := s.ownedAttribute->sortedBy(name);
 sPorts->collect(p| if(true) then {
  portsc.ownedAttribute += object Property{
   name:=p.name;
   visibility := VisibilityKind::public;
   _default := num.toString();
   type := numberType;
   isLeaf := true;
   isStatic := true;};
   num := num+1;
  } endif
 );
num:=0;
```

```
 }endif
 );
 var user: Set(Node)
   :=mo.getDeploymentDiagram().ownedElement[Node]
     ->select(n| n.hasStereotype("SecureMDD::User") or
   (not n
     .hasStereotype("SecureMDD::Terminal") and not n
       .hasStereotype("SecureMDD::Service") and not n
         .hasStereotype("SecureMDD::Smartcard")));
 user->collect(u| if(true) then{
  var uPorts: OrderedSet(Property)
    := u.ownedAttribute->sortedBy(name);
  uPorts->collect(p| if(true) then {
   portsc.ownedAttribute += object Property{
    name:=p.name;
    visibility := VisibilityKind::public;
    _default := num.toString();
    type := numberType;
    isLeaf := true;
    isStatic := true;};
    num := num+1;
   } endif
  );
 num:=0;
 }endif
 );
 return;
}

helper generateCardInitializeMethod
  ( inout mo : Model, inout card : Class) : Operation {
 var initMethod : Operation := object Operation{
    name:='initCard_'+card.name;
    type:=null;
    visibility:=VisibilityKind::public;
    isStatic:=false;
    ownedParameter := object Parameter{name
      :='port'; type:=getDataTypePrimitive(mo,'int');};
   };
 if(targetServer(mo))then{
 var attrs : OrderedSet(Property)
   := card.getAllAttributesForInitialization()
     ->asOrderedSet()->sortedBy(name);
 log("start");
 attrs->collect(c|if(true) then {
  log("initattr: " + c.name + " and type: " + c.type
    .name + " and transltype: " + c.type
      .translateType().name);
  c.getAppliedStereotypes()
    ->collect(s|log(" and stereotype: " + s
      .qualifiedName));
 } endif);
 card.getAllAttributes()->collect(c|if(true) then {
  log("real attr: " + c.name + " and type: " + c.type
```

```
    .name + " and transltype: " + c.type
      .translateType().name);
  c.getAppliedStereotypes()
    ->collect(s|log(" and stereotype: " + s
      .qualifiedName));
 } endif);
 while(not attrs->isEmpty()){
  var att : Property := attrs->first();
  initMethod.ownedParameter +=
   object Parameter{name:=att.name; type
     :=att.type.translateType();};
  attrs := attrs->reject(e|e=att)->asOrderedSet()
    ->sortedBy(name);
 }; }endif;
 if(targetTerminal(mo))then{
 var attrs : OrderedSet(Property)
   := card.getAllAttributesForInitialization();
 var i : Integer := 1;
 while(i <= attrs->size()){
  var att : Property := attrs->at(i);
  initMethod.ownedParameter +=
   object Parameter{name:=att.name; type
     :=att.type.translateType();};
  i := i+1;
 }; }endif;
 return initMethod;
}

helper removeHashedStereotype
  (inout source : Property) : Property
{
 source.unapplyStereotype(source
   .getAppliedStereotype('SecureMDD::Hashed'));
 return source;
}

helper unapplyStereotypes(inout model : Model){
 model.getClassDiagramClasses()
   ->forEach(source) { unapplyStereotypes4One
     (source); };
 return;
}

helper unapplyStereotypes4One(inout source : Class) {
  var st : Stereotype
    := source
      .getAppliedStereotype('SecureMDD::PlainData');
  if (st <> null) then { source
    .unapplyStereotype(st); } endif;
  st := source.getAppliedStereotype('SecureMDD::data');
  if (st <> null) then { source
    .unapplyStereotype(st); } endif;
  st
    := source
```

```
      .getAppliedStereotype('SecureMDD::HashData');
  if (st <> null) then { source
    .unapplyStereotype(st); } endif;
  st
    := source
      .getAppliedStereotype('SecureMDD::SignData');
  if (st <> null) then { source
    .unapplyStereotype(st); } endif;
}

helper insertDefaultArraysOperation(inout mo:Model){
var source: Class:=mo.getClassDiagramClasses()
  ->any(c|c.name='Arrays');
var iCodeable : Interface
  := mo.getClassDiagram().ownedElement[Interface]
    ->any(i | i.name='Codeable');
 source.ownedOperation+=object Operation{
 name:='equals';
  visibility:=VisibilityKind::public;
  isStatic:=true;
  type:=getDataTypePrimitive(mo,'boolean');
  var e1 : Parameter := object Parameter{name
    := 'left'; type
      := getDataTypePrimitive(mo,'boolean');upper
        :=-1; lower=0;};
  var e2 : Parameter := object Parameter{name
    := 'right'; type
      := getDataTypePrimitive(mo,'boolean');upper
        :=-1; lower=0;};
  ownedParameter+=e1;
  ownedParameter+=e2;};
  if(targetServer(mo))then {
   source.ownedOperation+=object Operation{
   name:='equals';
   visibility:=VisibilityKind::public;
   isStatic:=true;
   type:=getDataTypePrimitive(mo,'boolean');
   var e3 : Parameter := object Parameter{name
     := 'left'; type
       := getDataTypePrimitive(mo,'String');};
   var e4 : Parameter := object Parameter{name
     := 'right'; type
       := getDataTypePrimitive(mo,'String');};
   ownedParameter+=e3;
   ownedParameter+=e4;};
  }endif;
  source.ownedOperation+=object Operation{
  name:='equals';
  visibility:=VisibilityKind::public;
  isStatic:=true;
  type:=getDataTypePrimitive(mo,'boolean');
  var e5 : Parameter := object Parameter{name
    := 'left'; type
      := getDataTypePrimitive(mo,'byte');upper
```

```
        :=-1;lower:=0;};
  var e6 : Parameter := object Parameter{name
    := 'right'; type
      := getDataTypePrimitive(mo,'byte');upper
        :=-1; lower:=0;};
  ownedParameter+=e5;
  ownedParameter+=e6;};
  if (targetCard(mo))then{
  source.ownedOperation+=object Operation{
  name:='equals';
  visibility:=VisibilityKind::public;
  isStatic:=true;
  type:=getDataTypePrimitive(mo,'boolean');
  var e7 : Parameter := object Parameter{name
    := 'left';  type
      := getDataTypePrimitive(mo,'short');upper
        :=-1;lower:=0;};
  var e8 : Parameter := object Parameter{name
    := 'right'; type
      := getDataTypePrimitive(mo,'short');upper
        :=-1; lower:=0;};
  ownedParameter+=e7;
  ownedParameter+=e8;};}endif;
  source.ownedOperation+=object Operation{
  name:='equals';
  visibility:=VisibilityKind::public;
  isStatic:=true;
  type:=getDataTypePrimitive(mo,'boolean');
  var e9 : Parameter := object Parameter{name
    := 'left'; type := iCodeable ;upper:=-1;lower:=0;};
  var e10 : Parameter := object Parameter{name
    := 'right'; type := iCodeable;upper:=-1; lower
      :=0;};
  ownedParameter+=e9;
  ownedParameter+=e10;};
 if (targetCard(mo))then{
 source.ownedOperation+=object Operation{
  name:='copy';
  visibility:=VisibilityKind::public;
  isStatic:=true;
  var c1 : Parameter := object Parameter{name
    := 'from'; type
      := getDataTypePrimitive(mo,'byte');upper
        :=-1; lower=0;};
  var c2 : Parameter := object Parameter{name
    := 'to'; type
      := getDataTypePrimitive(mo,'byte');upper
        :=-1; lower=0;};
  ownedParameter+=c1;
  ownedParameter+=c2;};
  source.ownedOperation+=object Operation{
  name:='copy';
  visibility:=VisibilityKind::public;
  isStatic:=true;
```

```
  var c1 : Parameter := object Parameter{name
    := 'from'; type
      := getDataTypePrimitive(mo,'byte');upper
        :=-1; lower=0;};
  var c2 : Parameter := object Parameter{name
    := 'offset'; type
      := getDataTypePrimitive(mo,'short');};
  var c3 : Parameter := object Parameter{name
    := 'length'; type
      := getDataTypePrimitive(mo,'short');};
  var c4 : Parameter := object Parameter{name
    := 'to'; type
      := getDataTypePrimitive(mo,'byte');upper
        :=-1; lower=0;};
  ownedParameter+=c1;
  ownedParameter+=c2;
  ownedParameter+=c3;
  ownedParameter+=c4;};
 }endif;
 return;
}

query handleCertificateStereotype(in mo : Model)
{
 var cd : Package := mo.getClassDiagram();
 var cds : Set(Class) := mo.getClassDiagramClasses();
 var certclasses : Set(Class) := cds
   ->select(c | c
     .getAppliedStereotype
       ('SecureMDD::Certificate') <> null);
 certclasses
   ->collect(c | c
     .ownedOperation += object Operation{name
       :='verifyCertificate'; type
         :=getDataTypePrimitive
           (mo, 'Boolean'); ownedParameter+=object Parameter{name
             :='pubkey';type
               :=getSecureDataType(mo,"PublicKey");}});
 certclasses
   ->collect(c | generateCertificateOperation(mo,c));
 return;
}

query generateCertificateOperation
  (in mo : Model, inout cla : Class)
{
 var certificateDataClass : Class
   := cla.ownedAttribute
     ->select(a|a
       .getAppliedStereotype
         ('SecureMDD::signed') = null)
           ->any(true).type.oclAsType(Class);
 var paramsforGenerateCertificate : Sequence
   (Parameter) := certificateDataClass.ownedAttribute
```

```
      ->collect(a | object Parameter{name
        :=a.name; type:=a.type;lower:=a.lower;upper
          :=a.upper});
 var typePrivateKey : Class
   := mo.getSecurityDatatypes().oclAsType(Class)
     ->any(c|c.name = "PrivateKey");
 var parPrivateKey : Parameter
   := object Parameter{name:='issuerPrivateKey';type
     :=typePrivateKey;};
 cla.ownedOperation += object Operation{
 name:='generateCertificate';
 type:=cla;
 isStatic:=true;
 ownedParameter:=parPrivateKey;
 ownedParameter+=paramsforGenerateCertificate;
 };
 return;
}
```

2.5 Queries to transform a class diagram

This section contains several queries to execute some tests on the classes of the source class diagram as well as to select some elements of the class diagrams, e.g. a method that returns all classes which are codeable.

```
import swt.ModelExtensionLanguageQVTParser;
import commonQueries;
modeltype UML uses 'http://www.eclipse.org/uml2/3.0.0/UML';
library cdQueries access library commonQueries;

query Class::isSimpleCommClass() : Boolean {
 return self.getModel().getSecurityDatatypes()
   .ownedElement[Class]->exists(e|e=self);
}

query Class::isServer() : Boolean{
 var classes := self.getModel().getServers();
 var res := classes->exists(c | c.name = self.name);
 return res;
}

query Class::isUser() : Boolean{
 var classes := self.getModel().getUsers();
 var res := classes->exists(c | c.name = self.name);
 return res;
}

query Class::isTerminal() : Boolean{
 var classes := self.getModel().getTerminals();
 var res := classes->exists(c | c.name = self.name);
 return res;
}
```

```
query Class::isCard() : Boolean{
 var classes := self.getModel().getCards();
 var res := classes->exists(c | c.name = self.name);
 return res;
}

query Class::isStatefulServer() : Boolean{
 if(not self.isAbstract and self.isServer()) then{
  if(self.hasStatefulProperty()) then{
   return true
  }endif;
   if(self.general
     ->exists(c | c.hasStatefulProperty())) then{
   return true
  }endif;
 }endif;
 return false;
}

query Class::isList(): Boolean{
 return self.name.startsWith("ListOf");
}

query Class::hasMessageStereotype() : Boolean {
 return self.hasStereotype("SecureMDD::Message");
}

query hashUsed(in mo : Model) : Boolean
{
 var cds := mo.getClassDiagramClasses();
 return (cds->exists(c | c.ownedAttribute
   ->exists(a | a
     .getAppliedStereotype
       ('SecureMDD::hashed')<>null )));
}

query macUsed(in mo : Model) : Boolean
{
 var allMacA : Set(Class)
   := mo.getClassDiagramClasses()
     ->select(c | c.ownedAttribute
       ->exists(a | a
         .getAppliedStereotype
           ('SecureMDD::MAC') <> null));
 if(allMacA->size()>0) then
 {
  return true;
 }
 endif;
 return false;
}

query signUsed(in mo : Model) : Boolean
{
```

```
 var cds := mo.getClassDiagramClasses();
 var res :=  cds->exists(c | c.ownedAttribute
   ->exists(a | a
     .getAppliedStereotype
       ('SecureMDD::signed') <> null));
 return res;
}

query symEncUsed(in mo : Model) : Boolean
{
 var allSymEncA : Set(Class)
   := mo.getClassDiagramClasses()
     ->select(c | c.ownedAttribute
       ->exists(a | a
         .getAppliedStereotype
           ('SecureMDD::encrypted') <> null));
 if(allSymEncA->size()>0) then
 {
  return true;
 }
 endif;
 return false;
}

query asymEncUsed(in mo : Model) : Boolean
{
 var cds := mo.getClassDiagramClasses();
 var allSymEncA : Set(Class) := cds
   ->select(c | c.ownedAttribute
     ->exists(a | a
       .getAppliedStereotype
         ('SecureMDD::encryptedAsymm') <> null));
 if(allSymEncA->size()>0) then
 {
  return true;
 }
 endif;
 return false;
}

query Class::implementsInterface
  (i : String) : Boolean {
 return self.getAllImplementedInterfaces()
   ->exists(e|e.name.match(i));
}

query Class::isCodeable() : Boolean {
 if( self.superClass
   ->exists(e|e.isMessage()) or self
     .isMessage() ) then return true endif;
 if( self.isSimpleCommClass() ) then return true endif;
 if( self
   .isNeededCDClassForInitialization
     ()) then return true endif;
```

```
 if( self.hasStereotype('SecureMDD::HashData') or
  self.hasStereotype('SecureMDD::SignData') or
  self.hasStereotype('SecureMDD::MacData')  or
  self
    .hasStereotype
      ('SecureMDD::PlainData')) then return true endif;
 if( self.implementsInterface('HashData') or
  self.implementsInterface('SignData') or
  self.implementsInterface('MacData') or
  self
    .implementsInterface
      ('PlainData')) then return true endif;
 if( self.isList()) then return true endif;
 return self.getModel().getClassDiagramClasses()
   .ownedAttribute
     ->select(e|not (e = self) and e.type = self)
       ->exists(e|e._class.isCodeable());
}

query Model::getNeededCDClassesForInitialization
  () : Set(Class) {
 var initclas := self.getCDClassesForInitialization();
 var res := getReachableClasses(initclas);
 return res;
}

query Class::isNeededCDClassForInitialization
  () : Boolean{
 if( self.getModel().getCDClassesForInitialization()
  ->includes(self) ) then return true
 else {
  var cs : Set(Class)
    := getReachableClasses(self.getModel()
      .getCDClassesForInitialization());
  return cs->includes(self);
 } endif;
 return false;
}

query Class::isCDClassForInitialization() : Boolean{
 return self.getModel()
   .getCDClassesForInitialization()->includes(self);
}

query Class::isMessage() : Boolean {
 if( self
   .getAppliedStereotype
     ('SecureMDD::Message') <> null ) then return true endif;
 return self.superClass->exists(e|e.isMessage());
}

query Element::hasInitializeStereotype() : Boolean {
 return self.hasStereotype("SecureMDD::Initialize");
}
```

```
query Element::hasConstantStereotype() : Boolean {
 return self.hasStereotype("SecureMDD::Constant");
}

query Class::isPartOfAMessage() : Boolean {
 var mo : Model := self.getModel();
 return getReachableClasses(mo.getMessages())
   ->includes(self);

query getNotNecessaryElements(in mo : Model) : Set
  (Element)
{
 var cd : Package := mo.getClassDiagram();
 var associations : Set(Association)
   := cd.ownedElement[Association];
 var allTargetClasses: Set(Class)
   :=mo.getAllTargetClasses();
 var initclasses
   := mo.getNeededCDClassesForInitialization();
 var notNecessaryClasses : Set(Class)
   := cd.ownedElement[Class]->asSet()->reject(i |
       i.hasConstantStereotype()
   or allTargetClasses->includes(i)
   );
 if(targetTerminal(mo)) then
  notNecessaryClasses:= notNecessaryClasses
    ->reject(i| initclasses->includes(i))
 endif;
 var notNecessaryAssociations : Set(Element)
   := associations->select(e | notNecessaryClasses
     ->includesAll(e.memberEnd.type[Class]))[Element];
 return notNecessaryClasses[Element]
   ->union(notNecessaryAssociations);
}

query Model::getAllTargetClasses() : Set(Class) {
 var mo : Model :=self;
 var allTargetAcceptSendNodes : Set(ActivityNode)
   := mo.getActivities().ownedElement
     ->select(e|e.oclIsTypeOf(AcceptEventAction) or e
       .oclIsTypeOf(SendSignalAction))[ActivityNode]
  ->select(e|e.isInTargetPartition())->asSet();
 var allTargetAcceptSendClassnames
   := allTargetAcceptSendNodes
     ->collect(n | getSendReceiveClass(n.name));
 var allTargetMessages : Set(Class);
 if(targetServer(mo)) then
  allTargetMessages:= mo.getClassDiagramClasses()
    ->select(c|allTargetAcceptSendClassnames
      ->includes(c.name) or
   mo.getActivities().ownedElement[CallBehaviorAction]
     .ownedElement[Pin]
       ->exists(pin|pin.type.name=c.name))
 else
```

```
  allTargetMessages:= mo.getClassDiagramClasses()
    ->select(c|allTargetAcceptSendClassnames
      ->includes(c.name))
 endif;
 var allTargetClassesWithMessages: Set(Class)
   :=getReachableClasses(allTargetMessages
     ->union(mo.getTargetClasses()));
 return allTargetClassesWithMessages;
}

query ActivityNode::isInTargetPartition() : Boolean {
 return self.getModel().getTargetClasses()
   ->exists(t|t.name.match(self.inPartition
     ->any(true).getClassName()));
}

query Model::getClassDiagram() : Package {
 return self.ownedElement[Package]
   ->any(e|e
     .hasStereotype('SecureMDD::ClassDiagram'));
}

query Model::getSecurityDatatypes() : Package {
 return self.ownedElement[Package]
   ->any(e|e
     .hasStereotype('SecureMDD::SecurityDatatypes'));
}

query Model::getCodeableClasses():Set(Class){
 var codeableClasses: Set(Class)
   :=self
     .getClassDiagramClassesWithAttributeClasses()
       ->select(c | c.isCodeable());
 return codeableClasses
   ->union(self.getUsedSecurityDatatypes());
}

query Model::getClassDiagramClassesWithAttributeClasses
  ():Set(Class){
 return self.getClassDiagramClasses()
   ->union(self.getClassDiagramClasses()
     .ownedAttribute.type[Class]->asSet());
}

query Model::getClassDiagramClasses() : Set(Class) {
 return self.getClassDiagram().ownedElement[Class];
}

query Model::getUsedSecurityDatatypes():Set(Class){
 var usedSecurityDatatypes: Set(Class);
 if( hashUsed(self)) then
  usedSecurityDatatypes+=getSecureDataType
    (self,"HashedData")
 endif;
```

```
 if( signUsed(self)) then{
  usedSecurityDatatypes+=getSecureDataType
    (self,"SignedData")
 }endif;
 if (symEncUsed(self)) then{
  usedSecurityDatatypes+=getSecureDataType
    (self,"EncDataSymm");
  usedSecurityDatatypes+=getSecureDataType
    (self,"SymmKey");
 }endif;
 if( asymEncUsed(self)) then{
  usedSecurityDatatypes+=getSecureDataType
    (self,"EncDataAsymm");
  usedSecurityDatatypes+=getSecureDataType
    (self,"PublicKey");
  usedSecurityDatatypes+=getSecureDataType
    (self,"PrivateKey");
 }endif;
 if( macUsed(self)) then{
  usedSecurityDatatypes+=getSecureDataType
    (self,"MACData")
 }endif;
 return usedSecurityDatatypes;
}

query getSecureDataType
  (in mo : Model, name : String) : Class
{
 var packs := mo.ownedElement;
 var sd : Package := mo.getSecurityDatatypes();
 var dataType : Class := sd.ownedElement[Class]
   ->any(c | c.name = name).oclAsType(Class);
 return dataType;
}

query getClass(in mo : Model, name : String) : Class
{
 var dataType :=mo.getClassDiagramClasses()
   ->any(c|c.name=name);
 return dataType;
}

query Model::getTargetClasses(): Set(Class){
 var mo:Model:=self;
 if(targetCard(mo))then
  return mo.getCards()
 endif;
 if(targetTerminal(mo))then
  return mo.getTerminals()
 endif;
 if(targetServer(mo))then
  return mo.getServers()
 endif;
 return null;
```

```
}

query Model::getUsers() : Set(Class) {
 var classes : Set(Class) = self
   .getClassDiagramClasses();
 var users : Set(Class) := classes->select(e |
      e.hasStereotype('SecureMDD::User')
   or e.general
     ->exists(su | su.hasStereotype('SecureMDD::User'))
  );
 return users;
}

query Model::getCards() : Set(Class) {
 var classes : Set(Class) = self
   .getClassDiagramClasses();
 var smartcards : Set(Class) := classes->select(e |
  ( e.hasStereotype('SecureMDD::Smartcard')
   or e.general
     ->exists(su | su
       .hasStereotype('SecureMDD::Smartcard'))
  )
  and not (e.name =  'SimpleComm')
  );
 return smartcards;
}

query Model::getTerminals() : Set(Class) {
 var classes : Set(Class) = self
   .getClassDiagramClasses();
 var terminals : Set(Class) := classes->select(e |
      e.hasStereotype('SecureMDD::Terminal')
   or e.general
     ->exists(su | su
       .hasStereotype('SecureMDD::Terminal'))
   or e.hasStereotype('SecureMDD::PC')
   or e.general
     ->exists(su | su.hasStereotype('SecureMDD::PC'))
  );
 return terminals;
}

query Model::getServers() : Set(Class) {
 var classes : Set(Class) = self
   .getClassDiagramClasses();
 var servers : Set(Class) :=classes->select(e |
      e.hasStereotype('SecureMDD::Service')
   or e.general
     ->exists(su | su
       .hasStereotype('SecureMDD::Service'))
  );
 return servers;
}
```

```
query Class::getAllAttributesG() : Set(Property){
 if(self.general->isEmpty()) then
  return self.getAllAttributes()
  endif;
 return self.getAllAttributes()->union(self.general
   ->any(true).getAllAttributes())
}

query Class::getSubclasses() : Set(Class) {
 return self.getModel().getClassDiagramClasses()
   ->select(c | c.getGenerals()->includes(self));
}

query getDataTypePrimitive
  (in mo : Model, name : String) : PrimitiveType
{
 var sd : Package := mo.getSecurityDatatypes();
 var primType : PrimitiveType
   := sd.ownedElement[PrimitiveType]
     ->any(c | c.name = name);
 return primType;
}

query getInterface
  (in mo : Model, name : String) : Interface
{
 var intsWithName := mo.getAllInterfaces()
   ->select(i | i.name = name);
 if (intsWithName
   ->isEmpty()) then log
     ('##### ERROR in getInterface: Did not find ', name) endif;
 return intsWithName->any(true);
}

query Model::getAllInterfaces() :Set(Interface){
 return self.getClassDiagram().ownedElement[Interface];
}

query getEncAttr
  (inout source : Class, st : String) : Property
{
 return source.ownedAttribute
   ->any(a | a.getAppliedStereotype(st)<>null)
}

query Model::getSecurityDatatypesClasses() : Set
  (Class){
 return self.getSecurityDatatypes()
   .ownedElement[Class];
}

query getClassesRec(todo : Set(Class), done : Set
  (Class))  : Set(Class) {
 if( todo->isEmpty() ) then return done endif;
```

```
 var c : Class := todo->any(true);
 if( done
   ->includes(c) ) then return getClassesRec(todo
     ->excluding(c), done) endif;
 var newcs1 := c.ownedAttribute
   ->collect(p|p.type[Class]);
 var usages := c.clientDependency.supplier[Class];
 var newcs2 := c.general[Class];
 var todo2 : Set(Class) := object Set(Class){};
 var done2 : Set(Class) := object Set(Class){};
 todo2 += todo->excluding(c);
 todo2 += newcs1;
 todo2 += newcs2;
 todo2 += usages;
 done2 += done;
 done2 += c;
 return getClassesRec(todo2, done2);

query getReachableClasses(start : Set(Class))  : Set
  (Class) {
 var todo := start;
 var done : Set(Class) := Set { };
 while(todo->notEmpty()) {
  var c : Class := todo->any(true);
  todo := todo->excluding(c);
  if (done->includes(c)) then { } else {
   done += c;
   todo += c.ownedAttribute.type[Class];
   todo += c.clientDependency.supplier[Class];
   todo += c.general[Class];
  } endif;
 };
 return done;
}

query Model::getCDClassesForInitialization() : Set
  (Class) {
 var mo: Model := self;
 var cs : Set(Class) := object Set(Class){};
 if(targetServer(mo)) then{
 self.getServers()->collect(e|
 if(true) then{
  var initClasses: Set(Class)
    :=e.getCDClassesForInitialization();
   if(initClasses->notEmpty()) then{
    cs += initClasses
   }endif;
 }endif
 );}endif;
if(targetTerminal(mo) or targetCard(mo))then{
self.getCards()->collect(e|
 if(true) then{
  var initClasses: Set(Class)
    :=e.getCDClassesForInitialization();
```

```
    if(initClasses->notEmpty()) then{
     cs += initClasses
    }endif;
  }endif
  );
}endif;
 return cs;
}

query Class::getCDClassesForInitialization() : Set
  (Class) {
 return self.getCDAttributesForInitialization()
    .type[Class]->asSet();
}

query Class::getCDAttributesForInitialization() : Set
  (Property) {
 return self.getAllAttributes()
   ->select(e|e.hasInitializeStereotype() and self
     .getModel().getClassDiagramClasses()
       ->exists(x|e.type.oclAsType(Class)=x))->asSet();
}

query Class::getCDAttributesForInitialization() : Set
  (Property) {
 return self.getAllAttributesG()
   ->select(e|e.hasInitializeStereotype())->asSet();
}

query Class::getAllAttributesForInitialization
  () : OrderedSet(Property) {
 var attrs : OrderedSet(Property)
   := self.getAllAttributesG()
     ->select(e|e.hasInitializeStereotype())
       ->asOrderedSet()->sortedBy(name);
 if(self.getAllOperations()
   ->exists(c|c.name = self.name)) then {
  var attrs_const : OrderedSet(Parameter)
    := self.getAllOperations()
      ->any(c|c.name = self.name).ownedParameter;
  var isEqual : Boolean := true;
  attrs->collect(a|
   isEqual := isEqual.and(attrs_const
     ->exists(ac|ac.name = a.name and ac.type = a
       .type))
  );
  attrs_const->collect(a|
   isEqual := isEqual.and(attrs
     ->exists(ac|ac.name = a.name and ac.type = a
       .type))
  );
  if(isEqual) then {
   log
     ("Correct initialization constructor for class " + self
```

```
        .name + " found.");
    var attrs_new : OrderedSet(Property)
      := object OrderedSet(Property){};
    var i : Integer := 1;
    while(i <= attrs_const->size()){
     var att : Parameter := attrs_const->at(i);
     attrs_new += attrs
       ->any(c|c.name = att.name and c.type = att.type);
     log("init att: " + att.name);
     i := i+1;
    };
    attrs := attrs_new;
   }
   else {
    log
      ("Incorrect initialization constructor for class " + self
        .name + ", make sure it uses all and only initialize parameters
          . Destroying constructor.");
    self.getAllOperations()
      ->any(c|c.name = self.name).destroy();
   }endif;
 }endif;
 return attrs;
}

query Model::getAllSmartcardClasses() : OrderedSet
  (Class) {
 return getReachableClasses(self.getMessages()
   ->union(self.getCards()))->asOrderedSet()
     ->sortedBy(name);
}

query Model::getMessages() : Set(Class) {
 return self.getClassDiagramClasses()
   ->select(e | e.isMessage());
}

query Type::translateType() : Type {
 var mo : Model := self.getModel();
 if(self.oclIsTypeOf(Enumeration)) then {
  return getDataTypePrimitive(mo, 'byte');
 } endif;
 if(self.oclIsTypeOf(PrimitiveType)) then {
  return getDataTypePrimitive(mo, self
    .oclAsType(PrimitiveType).name);
 } endif;
 if(self.oclIsTypeOf(Class)) then {
  return self.oclAsType(Class);
 } endif;
 log("ERROR: unknown type!");
 return null;
}
```

2.6 Helper methods to transform a deployment diagram

This section contains auxiliary methods that are used to transform a (platform-independent) deployment diagram into the platform-specific deployment diagram. The methods introduced in this section are invoked from the transformations given in section 2.2 and 2.3.

```
import swt.ModelExtensionLanguageQVTParser;
import ddQueries;
import adQueries;
import cdQueries;
import commonQueries;
modeltype UML uses 'http://www.eclipse.org/uml2/3.0.0/UML';
library ddTransformations access library ddQueries,
        adQueries, cdQueries, commonQueries;

helper createDeploymentDiagram(inout mo:Model){
 if(mo.getDeploymentDiagram().ownedElement
  ->isEmpty()) then{
  log
    ("no deployment diagram found! create deployment diagram");
  var nodes : Set(Node):=object Set(Node){};
  var paths : Set(CommunicationPath);
  mo.getActivities().partition.getClassName()
    ->asSet()->collect(n|
   if(true) then{
    var node: Node:= object Node {name := n;};
    var clas: Class:=getClass(mo,n);
     if(clas.isServer()) then{
      node.applyStereotype(clas
        .getApplicableStereotype
          ("SecureMDD::Service"));
      if(clas.isStatefulServer()) then{
       node.getAppliedStereotype("SecureMDD::Service")
         .setProperty("stateful",true)
      }endif;
     }else{
       if(clas.isTerminal()) then{
       node.applyStereotype(clas
         .getApplicableStereotype
           ("SecureMDD::Terminal"))
      }else {
       if(clas.isCard()) then{
        node.applyStereotype(clas
          .getApplicableStereotype
            ("SecureMDD::Smardcard"))
       }endif
      }endif
     }endif
    nodes+=node;
  }endif
 );
 mo.getDeploymentDiagram().packagedElement+=nodes;
 var invokerControlFlows: Set(ControlFlow)
   :=mo.getActivities().ownedElement[ControlFlow]
     ->select(cf|cf.source.inPartition
```

```
        ->any(true)!=cf.target.inPartition
          ->any(true))->select(cf|not cf.source
            ->any(true).oclAsType(SendSignalAction)
              .isReturnMessage())->asSet();
 nodes->collect(n|
  if(true) then{
   var invokedNodes: Set(Node):=invokerControlFlows
     ->select(icf|icf.source.inPartition
       ->any(true).getClassName()=n.name).target
         .inPartition.getClassName().getNode(mo)
           ->asSet();
   invokedNodes->collect(iNode|
    if(true)then{
     var comPath: CommunicationPath
       := object CommunicationPath{
      var client: Property:=object Property{type
        :=n; visibility:=VisibilityKind::private; };
      var supplier: Property:=object Property{type
        :=iNode;visibility:=VisibilityKind::private;};
      ownedEnd+=client;
      ownedEnd+=supplier;
     };
     mo.getDeploymentDiagram()
       .packagedElement +=comPath;
     n.ownedAttribute+=comPath.memberEnd
       ->any(e|e.type.name=iNode.name);
    }endif
   );
  }endif
 );
 mo.getDeploymentDiagram().packagedElement+=paths;
}endif;
mo.getDeploymentDiagram().ownedElement[Node]
  ->forEach(n) {
 if (n.getAppliedStereotypes()->isEmpty()) then {
 var cla : Class := getClass(mo, n.name);
 if(cla.isTerminal()) then {
  n.applyStereotype(n
    .getApplicableStereotype("SecureMDD::Terminal"));
 }
 else if(cla.isServer()) then {
  n.applyStereotype(n
    .getApplicableStereotype("SecureMDD::Service"));
 }
 else if(cla.isCard()) then {
  n.applyStereotype(n
    .getApplicableStereotype("SecureMDD::Smartcard"));
 }
 else if(cla.isUser()) then {
  n.applyStereotype(n
    .getApplicableStereotype("SecureMDD::User"));
 }
 endif endif endif endif }
 endif;
```

```
  };
return;
}

helper createDefaultPortsInDeploymentDiagramm
  (inout mo:Model){
 var t2cPorts : Set(Property) := mo.getAllPorts()
   ->select(p|p.owner.oclAsType(Node)
     .hasStereotype("SecureMDD::Terminal")
   and p.type.oclAsType(Node)
     .hasStereotype("SecureMDD::Smartcard"))
 ;
 var num : Integer := 1;
 t2cPorts->collect(p|if(true) then{
  var cardNode : Node := p.type.name.getNode(mo);
  var termNode : Node
    := p.owner.oclAsType(Node).name.getNode(mo);
  if(getConnectedTerminalsForCard
    (termNode, cardNode) < getMaxConnectedTerminalsForCard
      (termNode, cardNode)) then {
   var c2tPort : Property := object Property{
    name := p.type.name.toFirstUpper()
     + '2'
     + termNode.name.toFirstUpper()
     + '_default'
     + num.toString();
    num := num+1;
    type := termNode
   };
   cardNode.ownedAttribute += c2tPort;
  } endif
 } endif);
 mo.getAllPorts()
   ->collect(p|if(p.name.oclIsUndefined() or p.name
     .match('')) then {
 p.name := p.owner.oclAsType(Node).name.toFirstUpper()
   + '2'
   + p.type.name.toFirstUpper()
   + '_default'
 } endif);
 return;
}
```

2.7 Helper methods to transform an activity diagram

This section contains auxiliary methods that are used to transform a (platform-independent) activity diagram into the platform-specific activity diagram. The methods introduced in this section are invoked from the transformations given in section 2.2 and 2.3.

```
import swt.ModelExtensionLanguageQVTParser;
import adQueries;
import cdQueries;
import ddQueries;
```

```
import commonQueries;
modeltype UML uses 'http://www.eclipse.org/uml2/3.0.0/UML';
library adTransformations access library adQueries,
                cdQueries, ddQueries, commonQueries;

helper partitionRepresentToPartitionName
  (inout mo: Model){
 mo.getActivities().partition->collect(p|
  if(p.name="") then
   p.name:=p.represents.oclAsType(Class).name
  endif
 );
 return;
}

helper clonePartition(inout mo:Model){
 var activities2 : Set(Activity):=mo.getActivities()
   ->select(a| a->partition->notEmpty());
 activities2->collect(a|generateDependencies(a));
 activities2:=mo.getActivities()->select(a| a
   ->partition->notEmpty());
 var found: Boolean:=true;
 while(found){
  found:=false;
  var createdActivities :Set(Activity)
    :=object Set(Activity){};
  activities2:=mo.getActivities()->select(a| a
    ->partition->notEmpty() and not a.isOperation());
  activities2->collect(activity|
   if(true) then{
    log("activityName: "+activity.name);
    var partitions: Set(ActivityPartition)
      := activity.partition;
    var upperClassPartition: ActivityPartition:=null;
     upperClassPartition:= partitions->any(p|
        p.isAbstractTargetComponent(mo) and
        p.getOwnedActivityNodes()
          ->select(n|n
            .oclIsTypeOf(AcceptEventAction) or n
              .oclIsTypeOf(InputPin)).incoming
                .source[SendSignalAction]
                  ->select(send|not send
                    .isReturnMessage()).inPartition
                      ->forAll(invokerPartition|
         not invokerPartition
           .isAbstractTargetComponent(mo))
    );
    if(not upperClassPartition.oclIsUndefined()) then{
     log("upperClassPartition: "+upperClassPartition
       .name);
     found:=true;
     var subClasses: Set(Class)
       :=getClass(mo,upperClassPartition
         .getClassName()).getSubclasses();
```

```
    var acceptEventActionsInUpperClassPartition: Set
      (AcceptEventAction)
        :=upperClassPartition
          .getOwnedActivityNodes()[AcceptEventAction];
    var invokerPartions: Bag(ActivityPartition)
      := acceptEventActionsInUpperClassPartition
        .incoming.source[SendSignalAction]->
       select(send|not send.isReturnMessage())
         .inPartition;
    var invokerPartionsWithSameStereotype : Bag
      (ActivityPartition):= invokerPartions
       ->select(p|p.isTargetComponent());
    var counter : Integer := 0;
    var hascloned : Boolean :=false;
    subClasses->collect(sc|
     if(invokerPartionsWithSameStereotype
      ->isEmpty
        () or invokerPartionsWithSameStereotype
         ->forAll(n| getNode(mo,n.getClassName())
           .getOutgoingNodes()
             ->exists(outgoingNode|outgoingNode
               .name=sc.name))) then {
      log("subClass: "+sc.name);
      hascloned:=true;
      var a: Activity
        := activity.deepclone().oclAsType(Activity);
      a.name:=a.name+counter.toString();
      counter:=counter+1;
      a.partition
        ->any(p| p.name=upperClassPartition.name)
          .name:=sc.name;
      createdActivities+=a;
     }endif
    );
    if(hascloned) then{
      activity.destroy();
    }endif;
   }endif;
  }endif
 );
 mo.getActivityDiagram()
   .packagedElement+=createdActivities;
 createdActivities
   ->forEach(a){generateDependencies(a);};
};
mo.getActivities()->forEach(a){
 log("activity: "+a.name);
 a.partition->collect(p|log("partition: "+p.name));
};
return;
}

helper generateDependencies(inout activity: Activity){
 var mo: Model:= activity.getModel();
```

```
 var sendSignalActions: Set(SendSignalAction);
 activity.partition->collect(p|
  if(true) then{
   sendSignalActions:=
    activity.ownedElement[SendSignalAction]
     ->select(s| isActivityNodeInPartition(s,p))
       ->asSet();
   sendSignalActions->collect(sendSignalAction|
    getNextAcceptActions(sendSignalAction,p)
     ->collect(acceptAction|
      activity.packagedElement+= object Dependency{
      client:=sendSignalAction;
      supplier:=acceptAction;
     sendSignalAction.createDependency(acceptAction)
    )
   )
  }endif
 );
 return;
}

helper handleActivities(inout mo:Model){
 log("Handling activity diagrams ...");
 var activities: Set(Activity):=mo.getActivities();
 handleSteps(mo,getAllTargetAcceptNodes(mo)
   ->reject(acc_node|acc_node.isPartOfOperation()));
 handleMethods(mo,(getAllOperations(mo)
   .getStartPoint())->asSet());
 activities.edge->select(e| e.source.inPartition
   ->any(true)!=e.target.inPartition->any(true) )
     ->collect(p|p.destroy());
 var partition: Bag(ActivityPartition)
   :=activities.partition
     ->select(p|not p.isTargetComponent());
 mo.getActivities()
   ->select(a|a.name="debit0").partition
     ->any(p|p.name=" aBank : AffiliatedBank")
       .getOwnedActivityNodes().name
         ->forEach(n){log("xx "+n)};
mo.getActivities()
  ->select(a|a.name="debit1").partition
    ->any(p|p.name=" aBank : AffiliatedBank")
      .getOwnedActivityNodes().name
        ->forEach(n){log("yy "+n)};
 partition+=getRedundantPartitions(mo);
 partition->node->collect(e|e.destroy());
 partition->edge->collect(e|e.destroy());
 partition->collect(p|p.destroy());
 activities->select(a|a.partition->isEmpty())
   ->collect(e|e.destroy());
 activities.ownedElement[ActivityEdge]
   ->select(e| e.target=null or e.source=null )
     ->collect(p|p.destroy());
 mo.getActivities().ownedElement[ActivityNode]
```

```
    ->collect(a|a.clearProcessed());
  mo.getActivityDiagram().ownedElement[Dependency]
    ->collect(d|d.destroy());
return;
}

helper handleSteps
  (inout mo: Model, inout acc_nodes : Set
    (ActivityNode)){
 acc_nodes->forEach(node){
  var cla : String := node.inPartition
    ->any(true).getClassName();
  setCurrentClass(cla);
  handleStep(mo,node);
 };
 return;
}

helper handleMethods
  (inout mo: Model, inout startPoints : Set
    (ActivityNode)){
 if(startPoints->notEmpty()) then{
  startPoints->forEach(node){
   var methodName : String
     := node.owner.oclAsType(Activity).name;
   setCurrentMethod(methodName);
   handleStep(mo,node);
  }
 }endif;
 return;
}

helper handleStep
  (inout mo : Model, inout node : ActivityNode ){
 if(node.isProcessed()) then {
  return
 } else {
  node.markProcessed()
 } endif;
 if(node.oclIsKindOf(AcceptEventAction)) then {
  handleAcceptEventAction(mo, node
    .oclAsType(AcceptEventAction));
 }endif;
 if(node.oclIsKindOf(DecisionNode)) then {
  mo.handleDecisionNode(node.oclAsType(DecisionNode));
 }endif;
 if(node.oclIsKindOf(SendSignalAction)) then {
  mo.handleSendSignalAction(node
    .oclAsType(SendSignalAction));
 }endif;
 if(node.oclIsKindOf(CallBehaviorAction)) then {
  node.name := MELAction2Code(node.name);
  node.outgoing->collect(n| handleStep(mo,n.target) );
 }endif;
```

```
}

helper ActivityNode::clearProcessed(){
 self.removeKeyword("PROCESSED");
 return
}

helper handleAcceptEventAction
  (inout mo: Model, inout node : AcceptEventAction){
  var edge_nxt : Set(ActivityEdge) := node.outgoing;
  if(MELReceiveAction2Code(node.name)<>'\n') then{
   var action: CallBehaviorAction
     := object CallBehaviorAction{
    name:=MELReceiveAction2Code(node.name);
    outgoing:=node.outgoing;
    node.outgoing:=null;
   };
   var edge: ActivityEdge:=object ControlFlow{
    source:=node;
    target:=action;
    weight:= object LiteralInteger{value:=1};
   };
   edge.inPartition+=node.inPartition;
   action.inPartition+=node.inPartition;
   node.owner.oclAsType(Activity).edge+=edge;
   node.owner.oclAsType(Activity).node+=action;
  }endif;
  if(targetServer(mo))then{
   var returnType: String
     := node.getModel().getClassDiagramClasses()
       ->select(e|e
         .hasStereotype("SecureMDD::Message"))
           ->any(true).name;
   node.name
     :='private '+returnType+" "+ getMethodString(node
       .oclAsType(AcceptEventAction),mo);
  }endif;
  if(targetTerminal(mo))then{
   node.name
     :='private void '+getMethodString(node
       .oclAsType(AcceptEventAction),mo);
  }endif;
  if(targetCard(mo))then{
   var c: Class
     := getClass(node
       .getModel(),getSendReceiveClass(node.name));
   if(c.isLeaf) then
    node.name
      :='public static void '+getMethodString(node
        .oclAsType(AcceptEventAction),mo)
   else
    node.name
      :='public void '+getMethodString(node
        .oclAsType(AcceptEventAction),mo)
```

```
    endif;
   }endif;
   edge_nxt->collect(n| handleStep(mo,n.target) );
   return;
}

helper Model::handleDecisionNode
   (inout node: DecisionNode){
  var mo:Model:=self;
  node.outgoing->collect(n|
   if(not (n.guard=null)) then{
     n.guard:= object OpaqueExpression{
      _body:=MELGuard2Code(n.guard.stringValue())
     }
   }endif
  );
  node.outgoing->forEach(n){
   handleStep(mo,n.target);
  };
  return;
}

helper Model::handleSendSignalAction
   (inout node: SendSignalAction){
  var mo:Model:=self;
  if(targetCard(mo))then{
   var s : String := MELSendAction2Code(node.name);
   node.name := s.substring(1,s.size());
  }endif;
  if(targetTerminal(mo))then{
   var targetPartitionName: String:= node.outgoing
     ->any(true).target.inPartition
       ->any(true).getClassName();
   var openSession : Boolean :=false;
   var closeSession : Boolean :=false;
   if(node.hasStereotype("SecureMDD::openSession")) then
    openSession:=true
   endif;
   if(node.clientDependency->any(true).target
     ->any(true).oclAsType(AcceptEventAction)
       .hasStereotype("SecureMDD::closeSession")) then
    closeSession:=true
   endif;
   if(node.oclAsType(SendSignalAction)
     .isReturnMessage() or node.name
       .find(" via ")>0 or node.name
         .find(" sendTo ")>0) then{
    node.name
      :=MELSendAction2Code(node
        .name,openSession,closeSession);
   }else{
    var partClassName := node.inPartition
      ->any(true).getClassName();
    var sendNode:Node
```

```
    :=getNode(node.getModel(),partClassName);
  var receiverClass: Class
    :=getClass(mo,node.outgoing
      ->any(true).target.inPartition
        ->any(true).getClassName());
  var port : String;
  if(not receiverClass.isAbstract) then{
   port := sendNode.ownedAttribute
     ->selectOne(p | p.type.name=receiverClass.name )
       .name;
  }else{
   port := sendNode.ownedAttribute
     ->selectOne(p | receiverClass.getSubclasses()
       ->exists(s|s.name=p.type.name) ).name;
  }endif;
  log("port: "+port);
  node.name
    := MELSendAction2Code(node
      .name+" via "+port,openSession,closeSession);
 }endif;
}endif;
if(targetServer(mo))then{
 var openSession : Boolean :=false;
 var closeSession : Boolean :=false;
 if(node.hasStereotype("SecureMDD::openSession")) then
  openSession:=true
 endif;
 if(node.clientDependency->any(true).target
   ->any(true).oclAsType(AcceptEventAction)
     .hasStereotype("SecureMDD::closeSession")) then
  closeSession:=true
 endif;
 if(node.name.find(" via ")>0 or node.name
   .find(" sendTo ")>0 or node
     .oclAsType(SendSignalAction)
       .isReturnMessage()) then{
  node.name
    :="return "+MELSendAction2Code(node
      .name, openSession, closeSession);
 }else{
  var sendNode:Node
    :=getNode(node.getModel(),node.inPartition
      ->any(true).getClassName());
  var receiverNode: Node
    :=getNode(node.getModel(),node.outgoing
      ->any(true).target.inPartition
        ->any(true).getClassName());
  var port: String := node.getModel().getAllPorts()
    ->select(p| p.owner[Node]
      ->any(true).name=sendNode.name and p.type
        .name=receiverNode.name)->any(true).name;
  node.name
    := "return "+MELSendAction2Code(node
      .name+" via "+port, openSession, closeSession);
```

```
  }endif;
 }endif;
 if(node.isPartOfOperation() and node
    .clientDependency
      ->any(true).target[AcceptEventAction]
        ->notEmpty()) then
  handleStep(mo,node.clientDependency
    ->any(true).target[AcceptEventAction]->any(true))
 endif;
 return;
}

helper ActivityNode::markProcessed(){
 self.addKeyword("PROCESSED");
 return
}

helper setActivityParameterNodesAndPinsToPartition
  (inout mo:Model){
 mo.getActivities()
    .ownedElement[ActivityParameterNode]
      ->forEach(n|n.oclAsType(ActivityParameterNode)
        .inPartition->isEmpty()){
  var node : ActivityParameterNode
    := n.oclAsType(ActivityParameterNode);
  if(node.outgoing->notEmpty()) then
   node.inPartition:=node.outgoing.target.inPartition
  else
   if(node.incoming->notEmpty()) then
    node.inPartition:=node.incoming.source.inPartition
   else
    logging
      ("ERROR: ActivityParameterNode with name "+node
        .name+" has neither outgoing nor incomming edges")
   endif
  endif;
 };
 mo.getActivities().ownedElement[CallBehaviorAction]
    .ownedElement[Pin]->forEach(n|n.inPartition
      ->isEmpty()){
  var node : Pin := n;
  if(node.owner.oclIsTypeOf(CallBehaviorAction)) then{
   node.inPartition
     := node.owner.oclAsType(CallBehaviorAction)
       .inPartition
  }endif
 };
}
 mo.getActivities()->forEach(a|a.isOperation()){
  if(a.isServiceOperation()) then
   a.applyStereotype(a
     .getApplicableStereotype
       ("SecureMDD::serviceOperation"))
  endif;
```

```
  a.applyStereotype(a
    .getApplicableStereotype("SecureMDD::operation"));
 };

helper test(inout mo:Model){
 var a : Node
   := mo.getDeploymentDiagram().ownedElement[Node]
     ->any(name="Debitcard");
 a.applyStereotype(getClass(mo,"Terminal")
   .getApplicableStereotype("SecureMDD::Terminal"));
 return;
}
```

2.8 Queries to transform an activity diagram

In this section the queries that are used to transform a platform-independent activity diagram into platform-specific class diagrams are shown.

```
import swt.ModelExtensionLanguageQVTParser;
import commonQueries;
import cdQueries;
modeltype UML uses 'http://www.eclipse.org/uml2/3.0.0/UML';
library adQueries access library cdQueries, commonQueries;

query Activity::isOperation(): Boolean{
 return self.partition
   ->notEmpty() and not self.partition
     ->exists(p|p.isUser());
}

query Activity::isServiceOperation() : Boolean{
 return self.isOperation() and
  self.getModel().getActivities()
    .ownedElement[CallBehaviorAction]
      ->exists(a|a.behavior.name = self.name and a
        .ownedElement[Pin]->notEmpty());
}

query ActivityNode::isPartOfOperation(): Boolean{
 return self.owner.oclAsType(Activity).isOperation();
}

query getAllOperations(inout mo:Model):Set(Activity){
 return mo.getActivities()->select(a|a.isOperation());
}

query Activity::getStartPoint(): ActivityNode{
 var mo : Model := self.getModel();
 if(self.ownedElement[ForkNode]->notEmpty()) then{
  return self.ownedElement[ForkNode]
    ->any(true).outgoing->any(true).target;
 }endif;
 var inParams
```

```
    :=self.ownedElement[ActivityParameterNode]
      ->select(n|n.parameter
        .direction = ParameterDirectionKind::_in or n
          .parameter
            .direction=ParameterDirectionKind::_inout);
 if(inParams->size()=1)then{
  return inParams->any(true).outgoing
    ->any(true).target;
 }endif;
 if(self.ownedElement[InitialNode]->notEmpty())then{
  return self.ownedElement[InitialNode]
    ->any(true).outgoing->any(true).target;
 }endif;
 return null;
}

query ActivityNode::isInTargetPartition() : Boolean {
 return self.getModel().getTargetClasses()
   ->exists(t|t.name.match(self.inPartition
     ->any(true).getClassName()));
}

query getAllTargetAcceptNodes(inout mo : Model) : Set
  (AcceptEventAction){
 var nodes : Set(AcceptEventAction)
   := mo.getActivities()
     .ownedElement[AcceptEventAction]
 ->select(e|e.isInTargetPartition())->asSet();
 return nodes;
}

query getAllTargetSendNodes(inout mo : Model) : Set
  (SendSignalAction){
 var nodes : Set(SendSignalAction)
   := mo.getActivities().ownedElement[SendSignalAction]
 ->select(e|e.isInTargetPartition())->asSet();
 return nodes;
}

query SendSignalAction::isReturnMessage(): Boolean{
  return true
 endif;
 var mo : Model := self.getModel();
 var partition : ActivityPartition :=self.inPartition
   ->any(true) ;
 if(partition.isUser()) then{
  return false;
 }endif;
 if(partition.isCard()) then{
  return true;
 }endif;
 var receiver: ActivityNode:=self.outgoing.target
   ->any(true);
 if(partition.isTerminal()) then{
```

```
  if(receiver.inPartition->any(true).isUser()) then
   return true
  endif;
 }endif;
 if(partition.isService()) then{
  if(receiver.oclIsTypeOf(Pin)) then
   return false
  endif;
  var dependencies: Set(Dependency)
    :=mo.getDependencies(self);
  var sendSignalAction: SendSignalAction
    := dependencies->any(d|
   d.target->oclAsType(AcceptEventAction)
     ->any(true)=receiver
       .oclAsType(AcceptEventAction)).source
         ->any(true).oclAsType(SendSignalAction);
  if(self.inPartition
    ->any(true)=sendSignalAction.outgoing.target
      ->any(true).inPartition->any(true)) then
   return true
  endif;
  return false;
 }endif;
 return false;
}

query ActivityPartition::isUser() : Boolean{
 var mo : Model := self.getModel();
 if (mo.getUsers()
   ->exists(c|c.name = self.getClassName())) then {
  return true;
 } endif;
 return false;
}

query ActivityPartition::isCard() : Boolean{
 var mo : Model := self.getModel();
 if(mo.getCards()
   ->exists(c|c.name = self.getClassName())) then{
  return true
 }endif;
 return false;
}

query ActivityPartition::isTerminal() : Boolean{
 var mo : Model := self.getModel();
 if(mo.getTerminals()
   ->exists(c|c.name = self.getClassName())) then{
  return true
 }endif;
 return false;
}

query ActivityPartition::isService() : Boolean{
```

```
 var mo : Model := self.getModel();
 if(mo.getServers()
   ->exists(c|c.name=self.getClassName())) then{
  return true
 }endif;
 return false;
}

query ActivityPartition::isAbstractTargetComponent
  (mo : Model): Boolean{
 return self.isTargetComponent() and getClass(mo,self
   .getClassName()).isAbstract;
}

query ActivityPartition::isTargetComponent(): Boolean{
 var mo:Model:=self.getModel();
 if(targetServer(mo))then{
  return mo.getServers()
    ->exists(c|c.name=self.getClassName());
 }endif;
 if(targetTerminal(mo))then{
  return mo.getTerminals()
    ->exists(c|c.name=self.getClassName());
 }endif;
 if(targetCard(mo))then{
  return mo.getCards()
    ->exists(c|c.name=self.getClassName());
 }endif;
 return false;
}

query isActivityNodeInPartition
  (node: ActivityNode, partition: ActivityPartition): Boolean{
 return node.inPartition->exists(p|p = partition);
}

query ActivityNode::isInTargetPartition(): Boolean{
 var mo: Model:=self.getModel();
 if(targetServer(mo))then{
  return self.isInServerPartition()
 }endif;
 if(targetTerminal(mo))then{
  return self.isInTerminalPartition()
 }endif;
 if(targetCard(mo))then{
  return self.isInSmartcardPartition()
 }endif;
 return false;
}

query ActivityNode::isInSmartcardPartition
  () : Boolean {
 var claname := self.inPartition
   ->any(true).getClassName();
```

```
 if (self.inPartition->isEmpty()) then {
  return false;
 }
 endif;
 return self.getModel().getCards()
   ->exists(t|t.name = claname);
}

query ActivityNode::isInServerPartition() : Boolean {
 var claname := self.inPartition
   ->any(true).getClassName();
 if (self.inPartition->isEmpty()) then {
  return false;
 }
 endif;
 return self.getModel().getServers()
   ->exists(t|t.name = claname);
}

query ActivityNode::isInTerminalPartition() : Boolean {
 var claname := self.inPartition
   ->any(true).getClassName();
 if (self.inPartition->isEmpty()) then {
  return false;
 }
 endif;
 return self.getModel().getTerminals()
   ->exists(t|t.name = claname);
}

query Node::isStatefulServer() : Boolean{
 if(self.getValue(self
    .getAppliedStereotype
      ("SecureMDD::Service"),"stateful")
        .oclAsType(Boolean)) then{
  return true
 }endif;
 return false;
}

query ActivityPartition::isEqual
  (partition : ActivityPartition) : Boolean{
 return self.name=partition.name and
 self.getOwnedActivityNodes().name
   ->includesAll(partition.getOwnedActivityNodes()
      .name) and
 partition.getOwnedActivityNodes().name
   ->includesAll(self.getOwnedActivityNodes().name) and
 self.edge.guard->reject(g|g=null).stringValue()
   ->includesAll(partition.edge.guard
     ->reject(g|g=null).stringValue()) and
 partition.edge.guard->reject(g|g=null).stringValue()
   ->includesAll(self.edge.guard
     ->reject(g|g=null).stringValue()) ;
}
```

```
query ActivityNode::isProcessed() : Boolean{
 return self.hasKeyword("PROCESSED");
}

query Model::getDependencies(n: ActivityNode):Set
  (Dependency){
var dependencies: Set(Dependency)
  :=self.getActivityDiagram()
     .ownedElement[Dependency]->
    select(d|d.target.owner.oclAsType(NamedElement)=d
      .supplier.owner.oclAsType(NamedElement) and
      d.target.owner.oclAsType(NamedElement)
        ->any(true)=getActivity(n));
return dependencies;
}

query getActivity(n: ActivityNode): Activity{
 return n.owner.oclAsType(Activity);
}

query getNextAcceptActions
  (n: ActivityNode, partition: ActivityPartition) : Set
    (AcceptEventAction){
 var s: Set(AcceptEventAction);
 var nodes: Set(ActivityNode):=n.outgoing.target
   ->asSet();
 if(n.oclIsTypeOf(SendSignalAction) and nodes
   ->size()=1 and nodes
     ->any(true).oclIsTypeOf(Pin))then{
  return nodes
    ->any(true).oclAsType(Pin).owner
      .oclAsType(CallBehaviorAction).output
        ->any(true).outgoing
          ->any(true).target
            .oclAsType(AcceptEventAction)->asSet();
 }endif;
 nodes->collect(node|
  if(isActivityNodeInPartition
    (node,partition) and node
      .oclIsTypeOf(AcceptEventAction)) then
    return node.oclAsType(AcceptEventAction)->asSet()
  endif
 );
 nodes->collect(node |
  if(node.outgoing->isEmpty()=false) then
   s+=getNextAcceptActions(node,partition)
  endif
 );
 return s;
}

query ActivityPartition::getOwnedActivityNodes():Set
  (ActivityNode){
 return self.owner.ownedElement[ActivityNode]
```

```
   ->select(e| e.inPartition->exists(p|p=self));
}

query getAllInitialNodes(in mo : Model) : Set
  (InitialNode){
 var nodes : Set(InitialNode)
   := mo.getActivities().ownedElement[InitialNode]
     ->select(n|n.inPartition->notEmpty())->asSet();
 return nodes;
}

query getRedundantPartitions(mo: Model): Set
  (ActivityPartition){
 var partitions : Set(ActivityPartition)
   := mo.getActivityDiagram().ownedElement[Activity]
     .ownedElement[ActivityPartition]->asSet();
 var partitions2 : Set(ActivityPartition)
   := mo.getActivityDiagram().ownedElement[Activity]
     .ownedElement[ActivityPartition]->asSet();
 var redundantPartitions : Set(ActivityPartition)
   :=object Set(ActivityPartition){};
 partitions->collect(p1|
  if(redundantPartitions
    ->isEmpty() or not redundantPartitions
      ->exists(rp|rp.isEqual(p1))) then{
   partitions2->collect(p2|
    if(p1!=p2 and p1.isEqual(p2)) then{
     redundantPartitions+=p2;
    }endif
   )
  }endif
 );
 return redundantPartitions;
}

query getActivityNode
  (mo: Model, name: String): ActivityNode{
 return mo.getActivities().ownedElement[ActivityNode]
   ->any(n|n.name=name);
}

query getStructuredInitNodes(in mo : Model) : Set
  (StructuredActivityNode){
 return mo.getActivities()
   .ownedElement[StructuredActivityNode]->asSet();
}

query getActivityParameterNodes(in mo: Model) : Set
  (ActivityParameterNode){
 return mo.getActivities()
   .ownedElement[ActivityParameterNode]->asSet();
}
```

```
query getSubActivities(in mo: Model) : Set(Activity){
 return mo.getActivities()->select(a|a.isOperation());
}

query getMethodString
  (acceptAction : AcceptEventAction, mo : Model): String{
 var c: String:=getSendReceiveClass(acceptAction.name);
 if(targetServer(mo))then{
  return 'process'+c
    .toFirstUpper()+'
      ('+c+' inmsg) throws ServiceException';
 }endif;
 if(targetTerminal(mo))then{
  return 'process'+c
    .toFirstUpper()+'
      ('+c+' inmsg) throws TerminalException';
 }endif;
 if (targetCard(mo))then{
  return 'process'+c
    .toFirstUpper()+'('+c+' inmsg)'; }endif;
  return null;
}
```

2.9 Helper methods

This section contains the methods that prepare a platform-independent model for transformation as well as those transformations that transform the security datatypes into platform-specific data types.

```
import cdTransformations;
import adTransformations;
import ddTransformations;
import commonQueries;
import cdQueries;
import adQueries;
import ddQueries;
import preparePim;
modeltype UML uses 'http://www.eclipse.org/uml2/3.0.0/UML';
library cdTransformations access library cdQueries,
              ddQueries, commonQueries, preparePim;

helper transform(inout model: Model){
    log("-- clone Partitions that have abstract classes");
    clonePartition(model);
    log("-- handle usermessage stereotype");
    removeUserStereoType(model);
    log("-- remove all unused Manual Classes");
    log("-- remove usages to message and manual classes");
    log("-- change PrimitiveTypes");
    applyTypes(model);
    log("-- handle list objects");
    handleLists(model);
    logging("-- getNotNecessaryElements");
```

```
    var remove : Set(Element)
     :=getNotNecessaryElements(model);
    log("-- save current cd classes");
    var cds : Set(Class)
     := model.getClassDiagramClasses();
    log("-- generate all Interfaces");
    generateAllInterfaces(model);
    log("-- generate all classes that are not in the pim");
    log("-- associate classe who implements
              interfaces with that interfaces");
    handleAllInterfaces(model);
    log("-- modify hasheddata class");
    generateHashedDataClass(model);
    log("-- modify signeddata class");
    generateSignedDataClass(model);
    log("-- modify encryption classes");
    generateEncClasses(model);
    log("-- generate MACdata class");
    generateMACClass(model);
    log("-- handle ServiceStereotype");
    log('handleCertificateStereotype');
    handleCertificateStereotype(model);
    log("-- add constructors to the transitive
              closure of the message and the sc class");
    buildConstructors(model);
    log("-- handle activities");
    handleActivities(model);
    log("-- unapply data stereotypes");
    unapplyStereotypes(model);
    log("-- removeNotNecessaryElements");
    remove->forEach(i) {i.destroy(); };
}
```

2.10 Queries

This section contains several methods to query some information of the source model, e.g. if a UML element (given as String) is annotated with a stereotype or if the target model of the current transformation is a smart card or a terminal model.

```
import swt.ModelExtensionLanguageQVTParser;
modeltype UML uses 'http://www.eclipse.org/uml2/3.0.0/UML';
library commonQueries;
static query targetTerminal(in mo : Model) : Boolean{
 if (mo.ownedComment
   ->exists(c| c.body='Terminal')) then
 return true endif;
 return false;
}
static query targetServer(in mo : Model) : Boolean {
 if (mo.ownedComment->exists(c| c.body='Server')) then
 return true endif;
 return false;
}
```

```
static query targetCard(in mo : Model) : Boolean{
 if (mo.ownedComment->exists(c|c.body='Javacard')) then
 return true endif;
 return false;
}
static query ispim2psm_executed(in mo:Model):Boolean{
 if (mo.ownedComment->exists(c| c.body='pim2psm')) then
 return true endif;
 return false;
}
static query Element::hasStereotype
  (st : String) : Boolean {
 return self.getAppliedStereotype(st) <> null;
}
static query Element::hasStatefulProperty() : Boolean {
 var res := false;
 var st
   := self.getAppliedStereotype("SecureMDD::Service");
 if(st <> null and self.getValue(st,"stateful")
   .oclAsType(Boolean)) then { res := true; } endif;
 return res;
}
static query Stereotype::setProperty
  (pName:String, value:Boolean){
 self->setValue(self,pName,true);
 return;
}
static query String::toFirstUpper() : String{
 if(self.repr().size() > 1) then
  return self.repr().substring(1, 1).toUpper() + self
    .repr().substring(2, self.repr().size())
 endif;
 return self.repr().toLower();
}
static query String::toFirstLower() : String{
 if(self.repr().size() > 1) then
  return self.repr().substring(1, 1).toLower() + self
    .repr().substring(2, self.repr().size())
 endif;
 return self.repr().toLower();
}
static query Model::getSequenceDiagram() : Package{
 return self.ownedElement[Package]
   ->any(e|e
     .getAppliedStereotype
       ('SecureMDD::SequenceDiagram') <> null);
}
static query Model::getSequences() : Set(Interaction){
 return self.getSequenceDiagram()
   .ownedElement[Collaboration]
     .ownedElement[Interaction]->asSet();
}
static query isSecureMDDModel(in mo: Model) : Boolean {
 var packs := mo.ownedElement;
```

```
 var withCD := packs
   ->exists(p | p.oclIsTypeOf(Package) and
                p.getAppliedStereotype
                  ('SecureMDD::ClassDiagram') <> null);
 return withCD;
}
static query pp(in li : Collection(String)) : String {
 var res : String := '';
 li->forEach(x) { res := res + x + ', '};
 return res;
}

query ppList(in s : Bag
  (String), bet : String) : String {
 var li := s->asSequence();
 var res : String := li->first();
 li->subSequence(2, li->size())
   ->forEach(x) { res := res + bet + x};
 return res;
}

query ActivityPartition::getClassName() : String{
 var n := self.name;
 if(n.size() = 0) then {
  n := self.represents.oclAsType(Class).name;
 } endif;
 if(n.find(":") < 1 ) then
  return n
 endif;
 return n.substring(n.find(":") + 1, n.size()).trim()
}

query Model::getActivityDiagram() :Package{
 return self.ownedElement[Package]
   ->any(e|e
     .hasStereotype('SecureMDD::ActivityDiagram'));
}

query Model::getActivities() : Set(Activity){
 return self.getActivityDiagram()
   .ownedElement[Activity];
}
```

Chapter 3

Xpand transformations: Generation of executable code

This section contains the model-to-text transformation that are implemented in Xpand and Xtend.

3.1 Transforming a platform-specific Smart Card model to code

3.1.1 Transforming a class annotated with stereotype ≪Smartcard≫

```
≪IMPORT uml≫
≪EXTENSION psm_templates_shared::translations≫
≪EXTENSION psm_templates_shared::parse≫
≪EXTENSION psm_templates_shared::getter≫
≪EXTENSION psm_templates_shared::tests≫
≪EXTENSION psm_templates_shared::names≫

≪DEFINE main FOR Model≫
≪IF !this.ownedElement.isEmpty≫
 ≪FOREACH getCards().reject(e|e.isAbstract())
   AS card≫
  ≪setDirectoryPrefix(card)≫
  ≪EXPAND generateCard FOR card≫
  ≪logging("### generateCard done
    . Now generateStore. ###")≫
  ≪EXPAND generateStore FOR card≫
  ≪logging("### generateStore done
    . Now generateClasses ###")≫
  ≪EXPAND psm_templates_shared::classesShared
    ::generateClass FOREACH this
      .getNormalGeneratedClasses(card)≫
  ≪EXPAND generateGate≫
  ≪EXPAND generateDummyGate≫
  ≪EXPAND psm_templates_shared::classesShared
    ::generateManualClasses
      FOR this.getClassDiagram().ownedElement
        .typeSelect(Class).select(e|e
          .hasStereotype("SecureMDD::Manual"))≫
```

```
 «ENDFOREACH»
«ENDIF»
«ENDDEFINE»

«DEFINE generateCard FOR Class»
«FILE this.getModel().getDirectoryPrefix()
  +filename()-»
package «this.getModel().packagenameCard()-»;
 import javacard.framework.*;
 «EXPAND psm_templates_shared::classesShared
    ::generateImportManuals FOR this»
public «IF this
   .isAbstract
     ()-»abstract«ENDIF-» class «name-» extends «simpleCommName
       ()-» {
«FOREACH this.getAllAttributesG().reject(e|this
   .getModel().getClass('SimpleComm').ownedAttribute
     .contains(e) ) AS p-»
 «p.getAttributeImplCode()»
«ENDFOREACH-»
«IF !this.isAbstract()-»
 «EXPAND CardConstructor FOR this-»
 «EXPAND CardInitializer FOR this-»
 «EXPAND generateProcess FOR this-»
«ENDIF-»
«EXPAND psm_templates_shared::classesShared
   ::generateActivities FOR this-»
«EXPAND psm_templates_shared::classesShared
   ::generateManualMethods FOR this»
}
«ENDFILE»
«ENDDEFINE»

«DEFINE generateStore FOR Class»
«LET this.getModel().getClass(storeName())
  AS store-»
«FILE this.getModel().getDirectoryPrefix()
  + storeName() + ".java"»
package «this.getModel().packagenameCard()-»;
public class «storeName()» {
 «FOREACH store.instanceAttributes() AS e-»
  «e.getAttributeImplCode()»
 «ENDFOREACH-»
 «FOREACH store.ownedOperation AS o-»
 «LET o.name.subString(3,o.name.length) AS n-»
 «IF o.name!="initAll" && o.name!="reset" && o
    .name!=store.name-»
   «IF !o.ownedParameter.exists(e|e.direction
     .toString()=="in" || e.direction
       .toString()=="inout") »
     «IF isUsedCodeableClass(this.getModel()
       .getClass(n)) -»
      private static «n»[] «n»Array;
      private static byte «n»Count = 0;
```

```
      private static void init«n»() {
       «n»Array = new «n»[«n»MaxCountFor«this
         .name»];
       for(short i=0; i < «n»Array.length; i++)
        «n»Array[i] = new «n»();
      }
      public static «n» new«n»
        () { return «n»Array[«n»Count+
          +]; }
     «ELSE-»
      public static «n» new«n»
        () { return null; }
     «ENDIF-»
    «ELSE»
     «LET this.getModel().getClass(n) AS c»
      «IF isUsedCodeableClass(this.getModel()
        .getClass(n)) -»
      public static «n» new«n»
        («FOREACH c.getAttributesInCorrectOrder()
          AS attr SEPARATOR ", "»«attr
            .translateType()
              .translatePrimitiveType()» «attr
                .name»«ENDFOREACH») {
       «n» _«n» = new«n»();
       «FOREACH c.instanceAttributes() AS attr-»
        «IF attr.isPrimitiveType()-»
         _«n».«attr.name» = «attr
           .name»;
        «ELSEIF attr.isPrimitiveTypeArray()-»
         Arrays.copy(«attr.name», _«n»
           .«attr.name»);
        «ELSE-»
         _«n».«attr.name»
           .copy(«attr.name»);
        «ENDIF-»
       «ENDFOREACH-»
       return _«n»;
       }
      «ELSE-»
       public static «n» new«n»
         («FOREACH c.instanceAttributes()
           AS attr SEPARATOR ", "»«attr
             .translateType()
               .translatePrimitiveType()» «attr
                 .name»«ENDFOREACH») {
        return null;
       }
      «ENDIF-»
     «ENDLET»
    «ENDIF-»
  «ENDIF-»
«ENDLET-»
«ENDFOREACH-»
public static void initAll() {
```

```
 «FOREACH store.ownedOperation AS o-»
  «IF o.name!="initAll" && o.name!="reset" && o
    .name!=store.name && o.ownedParameter
      .size == 1-»
  «LET o.name.subString(3,o.name.length) AS n-»
«IF isUsedCodeableClass(this.getModel()
  .getClass(n)) -»
   init«n»();
«ENDIF-»
  «ENDLET-»
  «ENDIF-»
 «ENDFOREACH-»
 }
 public static void reset() {
 «FOREACH store.ownedOperation AS o-»
 «IF o.name!="initAll" && o.name!="reset" && o
   .name!=store.name && o.ownedParameter
     .size == 1-»
 «LET o.name.subString(3,o.name.length) AS n-»
 «IF isUsedCodeableClass(this.getModel()
   .getClass(n)) -»
 «n»Count = 0;
 «ENDIF-»
 «ENDLET-»
 «ENDIF-»
 «ENDFOREACH-»
 }
}
«ENDFILE»
«ENDLET-»
«ENDDEFINE»

«DEFINE generateGate FOR Model»
«logging("Generating class Gate")»
 «FILE this.getDirectoryPrefix()+gateName()
   + ".java"-»
  package «this.getModel().packagenameCard()-»;
  import javacard.framework.APDU;
  public interface «gateName()» {
  public void callConstraints();
  public void sendAPDU
    (SimpleComm card, APDU apdu, byte[] data
      , short remLen, boolean first);
  public void finishProtocol(SimpleComm card);
  }
 «ENDFILE»
«ENDDEFINE»

«DEFINE generateDummyGate FOR Model»
«logging("Generating class DummyGate")»
 «FILE this.getDirectoryPrefix() + "Dummy"
   + gateName() + ".java"-»
  package «this.getModel().packagenameCard()-»;
  import javacard.framework.APDU;
```

```
  public class Dummy«gateName
    ()» implements «gateName()»{
  public void callConstraints() {
  }
  public void sendAPDU
    (SimpleComm card, APDU apdu, byte[] data
      , short rem_len, boolean first) {
   card.sendAPDU(apdu, data, rem_len, first);
  }
  public void finishProtocol(SimpleComm card){
  }
  }
 «ENDFILE»
«ENDDEFINE»

«DEFINE CardConstructor FOR Class»
 public «name»() {
  «FOREACH getAllAttributesG().reject(e|e
    .isCustom(this)) AS attr-»
   «IF !attr.isLeaf-»
    «IF attr.isStateAttribute()-»
     «attr.name»=«attr.type.name»
       .APPLET_UNINITIALIZED;
    «ELSE-»
     «attr.name»=«attr.initialValue()»;
    «ENDIF-»
   «ENDIF-»
  «ENDFOREACH-»
 }
«ENDDEFINE»

«DEFINE CardInitializer FOR Class»
 public void appletInitialization
   (byte[] buf, short offset) {
  if («this.getStateAttribute()
    .name» != «this.getStateAttribute().type
      .name»
        .APPLET_UNINITIALIZED) «abortName()»();
  «codingName()».getInstance().decode«this
    .name»(buf, offset, this);
 }
  «LET this.instanceOperations().select(e|e.name
    .toString()==this.name.toString()) AS consops-»
  «FOREACH consops AS op-»
   «IF op.ownedParameter.size>0-»
   public «this.name
     .toString()» init(«FOREACH op
       .ownedParameter
         AS param SEPARATOR ","»«op.class
           .getPropertyForParam(param)
             .translateType()» «param
               .name»«ENDFOREACH») {
     «FOREACH op.ownedParameter AS param-»
      this.«op.class.getPropertyForParam(param)
```

```
        .name≫=≪param.name≫;
      ≪ENDFOREACH-≫
      ≪FOREACH getAllAttributesG().select(e|e
        .hasInitValue()).reject(e|op.ownedParameter
          .name.contains(e.name) || e
            .isLeaf || getCustomCardClassAttributes()
              .contains(e.name)) AS attr-≫
       ≪attr.name≫=≪attr
         .initialValue()≫;
      ≪ENDFOREACH-≫
      return this;
     }
    ≪ENDIF-≫
   ≪ENDFOREACH-≫
   public ≪this.name.toString()≫ init(){
    ≪FOREACH getAllAttributesG().select(e|e
      .hasInitValue()).reject(e|e
        .isLeaf || getCustomCardClassAttributes()
          .contains(e.name)) AS attr-≫
     ≪attr.name≫=≪attr.initialValue()≫;
    ≪ENDFOREACH-≫
    return this;
   }
  ≪ENDLET-≫
≪ENDDEFINE≫

≪DEFINE generateProcess FOR Class≫
 public void process(≪this.getModel()
   .getNameMessageSuperclass()≫ inmsg) {
  if (≪this.getStateAttribute()
    .name≫ == ≪this.getStateAttribute().type
      .name≫
        .APPLET_UNINITIALIZED) ≪abortName()≫();
  currentGate.callConstraints();
  switch(inmsg.getCode()) {
  ≪FOREACH this.getAllAcceptEventActions()
    AS signal-≫
   case ≪codeName()≫.≪signal.name
     .toString().getSendReceiveClass()
       .toUpperCase()-≫
   : process≪signal.name.toString()
     .getSendReceiveClass()
       .toFirstUpper()-≫
         (≪EXPAND generateMethodParams
           FOR (AcceptEventAction)signal-≫);
    break;
  ≪ENDFOREACH-≫
   default: ≪abortName()≫();
  }
  currentGate.callConstraints();
 }
≪ENDDEFINE≫

≪DEFINE generateMethodParams
```

```
  FOR AcceptEventAction-≫
(≪this.name.toString()
   .getSendReceiveClass()≫) inmsg≪ENDDEFINE≫

≪DEFINE getMethodString FOR AcceptEventAction-≫
 ≪LET this.getLabel().toString()
    .getSendReceiveClass() AS Cla-≫
  ≪Cla.toFirstLower()≫(≪Cla≫ inmsg)
 ≪ENDLET-≫
≪ENDDEFINE≫
```

3.1.2 Generating the Coding for the smart cards

```
≪IMPORT uml≫
≪EXTENSION psm_templates_shared::translations≫
≪EXTENSION psm_templates_shared::getter≫
≪EXTENSION psm_templates_shared::tests≫
≪EXTENSION psm_templates_shared::names≫
≪EXTENSION psm_templates_shared::parse≫

≪DEFINE main FOR uml::Model≫
≪IF !this.ownedElement.isEmpty≫
≪logging("started ...")≫
≪FOREACH getCards().reject(e|e.isAbstract())
  AS card≫
 ≪setDirectoryPrefix(card)≫
 ≪FILE getDirectoryPrefix()+codingName()
   +".java"≫
 ≪logging("file is " + getDirectoryPrefix()
   +codingName()+".java")≫
  package ≪this.getModel().packagenameCard()-≫;
  ≪LET packagedElement.typeSelect(Package)
    AS packages≫
   ≪LET this.getCodeableClasses(card).reject(c|c
     .isAbstract()) AS classes≫
    ≪EXPAND psm_templates_shared::codingShared
      ::generateCodingHeader≫
    ≪EXPAND psm_templates_shared::codingShared
      ::generateCodingMain(this.getModel())
        FOR classes≫
    ≪EXPAND generateCodingForClass FOREACH classes≫
    ≪EXPAND generateCoding4Card FOR card≫
    ≪EXPAND psm_templates_shared::codingShared
      ::generateCodingFooter≫
   ≪ENDLET≫
  ≪ENDLET≫
 ≪ENDFILE≫
≪ENDFOREACH≫
≪ENDIF≫
≪ENDDEFINE≫
```

3.1.3 Generating the class SimpleComm and other classes for the smart cards

```
«IMPORT uml»
«EXTENSION psm_templates_shared::getter»
«EXTENSION psm_templates_shared::names»
«EXTENSION psm_templates_shared::tests»
«EXTENSION psm_templates_shared::parse»
«EXTENSION psm_templates_shared::translations»

«DEFINE main FOR uml::Model»
«IF !this.ownedElement.isEmpty»
«FOREACH getCards().reject(e|e.isAbstract())
  AS card»
 «setDirectoryPrefix(card)»
 «EXPAND psm_templates_shared::simplecommShared
   ::generateSimplecommClasses FOR card»
«ENDFOREACH»
«ENDIF»
«ENDDEFINE»
```

3.2 Transforming a platform-specific Terminal model to code

3.2.1 Transforming a class annotated with stereotype «Terminal»

```
«IMPORT uml»
«EXTENSION psm_templates_shared::translations»
«EXTENSION psm_templates_shared::parse»
«EXTENSION psm_templates_shared::getter»
«EXTENSION psm_templates_shared::tests»
«EXTENSION psm_templates_shared::names»

«DEFINE main FOR Model»
 «FOREACH getTerminals().reject(e|e
   .isAbstract()) AS terminal»
  «setDirectoryPrefix(terminal)»
  «EXPAND psm_templates_shared::classesShared
    ::generatePorts»
  «EXPAND generateGate FOR terminal»
  «EXPAND psm_templates_shared::classesShared
    ::generateLenghtConstants»
  «IF this.hasUserMessage()»
   «EXPAND generateUserinterface»
   «EXPAND generateUserMessage FOREACH this
     .getUserMessageClasses(terminal)»
  «ENDIF»
  «IF !terminal.getDirectlyUsedServers()
    .isEmpty»
   «EXPAND psm_templates_shared::classesShared
     ::generateMessageWrapper»
  «ENDIF»
  «EXPAND generateTerminal FOR terminal»
  «EXPAND psm_templates_shared::classesShared
    ::generateClass FOREACH this
      .getNormalGeneratedClasses(terminal)»
  «EXPAND psm_templates_shared::classesShared
    ::generateException»
```

```
  «EXPAND psm_templates_shared::classesShared
    ::generateDumpStubs FOREACH terminal
      .getDirectlyUsedServers()»
  «EXPAND psm_templates_shared::classesShared
    ::generateDumpServiceIdentifierFile
     FOR terminal»
  «IF !terminal.getDirectlyUsedServers()
    .isEmpty»
   «EXPAND psm_templates_shared::classesShared
     ::generateStubsGenerator FOR terminal»
  «ENDIF»
 «ENDFOREACH»
 «EXPAND psm_templates_shared::classesShared
   ::generateManualClasses
    FOR this.getClassDiagram().ownedElement
      .typeSelect(Class).select(e|e
        .hasStereotype("SecureMDD::Manual"))»
«ENDDEFINE»

«DEFINE generateTerminal FOR Class»
«debug("Generating Terminal "
  +this.name.toString())-»
 «FILE this.getModel().getDirectoryPrefix()
   +filename()-»
  package «this.getModel().packagename()-»;
  import swt.terminal.HandleResult;
  import swt.util.ByteArray;
  import swt.util.HexString;
  import de.uniaugsburg.swt.javacard.terminalinterface
    .SWTReader;
  import de.uniaugsburg.swt.javacard.terminalinterface
    .SWTReaderException;
  import javax.xml.ws.BindingProvider;
  «EXPAND psm_templates_shared::classesShared
    ::generateImportStubs FOR this»
  «EXPAND psm_templates_shared::classesShared
    ::generateImportManuals FOR this»
  public class «name-»{
   «EXPAND psm_templates_shared::classesShared
     ::generateAllAttributes FOR this»
   «EXPAND psm_templates_shared::classesShared
     ::generateStubsAttributes FOR this»
   «EXPAND generateAttributesAndMethodsForUserConnection
     FOR this»
   «EXPAND generateAttributesAndMethodsForGates
     FOR this»
   «EXPAND psm_templates_shared::classesShared
     ::Constructor FOR this-»
   «EXPAND psm_templates_shared::classesShared
     ::genrateSetTimeoutMethod FOR this»
   «EXPAND psm_templates_shared::classesShared
     ::generateMethodsForInvokingMultipleInstantiatedServices
      FOR this»
   «EXPAND psm_templates_shared::classesShared
```

```
    ::generateSendMsgMethods FOR this≫
  ≪EXPAND generateProcessMethods FOR this≫
  ≪EXPAND generateAttributesAndMethodsForCardConnections
    FOR this≫
  ≪EXPAND generateInitCardMethods FOR this≫
  ≪EXPAND psm_templates_shared::classesShared
    ::generateActivities FOR this≫
  ≪EXPAND psm_templates_shared::classesShared
    ::Getter FOREACH this.getAllAttributesG()-≫
  ≪EXPAND psm_templates_shared::classesShared
    ::Setter FOREACH this.getAllAttributesG()-≫
  ≪EXPAND psm_templates_shared::classesShared
    ::generateManualMethods FOR this≫
  ≪EXPAND psm_templates_shared::classesShared
    ::generateExceptionMehtod FOR this≫
 }
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateProcessMethods FOR Class≫
 public void processRawMessage(byte[] data) {
  prevMsg = data;
  try{
   Coding.getInstance().decodeInit();
   Codeable cResponse = Coding.getInstance()
     .decode(data);
   processMessage((≪this.getModel()
     .getNameMessageSuperclass()≫) cResponse);
  } catch(java.lang.Exception e) {
   System.err
     .println
       ("Failed decoding data in processRawMessage");
  }
 }
 private void processMessage(≪this.getModel()
   .getNameMessageSuperclass()≫ inmsg)
     throws ≪exceptionName()≫ {
  currentGate.callConstraints();
  switch(inmsg.getCode()) {
  ≪FOREACH this.getAllAcceptEventActions()
    AS signal-≫
   case ≪codeName()≫.≪signal.name
     .toString().getSendReceiveClass()
       .toUpperCase()-≫:
    process≪signal.name.toString()
      .getSendReceiveClass()
        .toFirstUpper()-≫((≪(
          (AcceptEventAction)signal).name.toString()
            .getSendReceiveClass()≫) inmsg);
   break;
  ≪ENDFOREACH-≫
   default: ≪abortName()≫();
   }
   currentGate.callConstraints();
```

```
    currentGate.finishProtocol(this);
   }
   public void process(≪this.getModel()
     .getNameMessageSuperclass()≫ msg)
       throws ≪terminalExceptionName()≫{
  currentGate.callConstraints();
  try{
   if(msg instanceof ≪this.getModel()
     .getUserMessage().name≫)
    prevUMsg = (≪this.getModel().getUserMessage()
      .name≫)msg;
   processMessage(msg);
  } catch (java.lang.Exception e) {
   ≪FOREACH this.getStructuredInitNodes()
     .select(e|e
       .hasStereotype("SecureMDD::Exception"))
         AS exceptionMethod≫
   ≪IF exceptionMethod.inPartition.first()
     .getPartitionClass()==this.name
       .toString() || ((! this.superClass
         .isEmpty)?(exceptionMethod.inPartition
           .first().getPartitionClass()==this
             .superClass.first().name
               .toString()):
                 (false)) || exceptionMethod.owner
                   .metaType==StructuredActivityNode-≫
   ≪exceptionMethod.name≫();
   ≪ENDIF≫
   ≪ENDFOREACH-≫
   if(e instanceof ≪terminalExceptionName()≫)
    throw (≪terminalExceptionName()≫)e;
   else
    throw new ≪terminalExceptionName()≫
     ("Error when calling process method: " + e);
  }
  currentGate.callConstraints();
  currentGate.finishProtocol(this);
 }
≪ENDDEFINE≫

≪DEFINE generateInitCardMethods FOR Class≫
≪FOREACH this.ownedOperation.select(e|e.name
  .startsWith('initCard_')) AS op-≫
 public void ≪op.name≫(≪FOREACH op
   .ownedParameter
     AS param SEPARATOR ", "-≫≪param.type.name
       .toString()≫ ≪param.name
         .toString()≫≪ENDFOREACH-≫)
           throws ≪terminalExceptionName()≫ {
 ≪IF this.ownedOperation.select(e|e.name
   .startsWith('initCard_')).reject(e|e.name
     .matches(this.name)).size > 1-≫
  byte cardCode = 0x0;
  switch(port) {
```

```
 «FOREACH getModel().getTerminal2CardPorts()
    .select(p|((Node)p.owner).name==this.name)
      AS port-»
 case «deploymentPropertiesName()»
    .«port.name»:
  cardCode = «codeName()».«port.type
    .name.toString().toUpperCase()»;
  break;
 «ENDFOREACH-»
 default:
  «abortName()»();
  break;
}
«ELSE-»
 byte cardCode = «codeName()».«op.name
    .toString().split('_').get(1)
      .toUpperCase()-»;
«ENDIF-»
byte[] apdu = { 0, 4, 0, 0, 0, cardCode};
«FOREACH op.ownedParameter.withoutFirst()
 AS attr-»
«IF attr.translateType()=="String"-»
  if (!«codingName()».getInstance()
    .isValidCardString(«attr.name»))
   System.err.println("Encoded string "
     + «attr.name» + " is longer than "
       + «lengthConstantsName()»
         .LENGTHOFSTRING + " bytes.");
«ENDIF-»
apdu = ByteArray.append(apdu,
«IF attr.translateType()
  .isPrimitiveTranslatedType() || attr
    .isPrimitiveType()-»
 «IF attr.translateType()=="short" || attr
   .translateType()=="int"-»
  ByteArray.append(«codeName()»
    .INT, ByteArray.toByteArray(«attr
      .name»)));
 «ENDIF-»
 «IF attr.translateType()=="String"-»
  («attr
    .name» != null) ? «codingName()»
      .getInstance().encodeSingleString(«attr
        .name») : new byte[]{«codeName
          ()».IGNORE} );
 «ENDIF-»
«IF attr.translateType()=="boolean" || attr
  .translateType()=="byte"-»
  new byte[]{«codeName()».«attr
    .translateType().toString()
      .toUpperCase()», «attr.name»});
 «ENDIF-»
«ELSE-»
 («attr.name» != null) ? encode(«attr
```

```
      .name≫) : new byte[]{≪codeName()≫
        .IGNORE});
   ≪ENDIF-≫
  ≪ENDFOREACH-≫
   try {
   transmit(apdu, port);
  } catch (SWTReaderException e) {
   ≪abortName()≫();
  }
 }
≪ENDFOREACH-≫
≪ENDDEFINE≫

≪DEFINE generateAttributesAndMethodsForCardConnections
  FOR Class≫
  private final static int MAX_SPLIT_LENGTH = 50;
  private final static int MAX_APDU_LENGTH = 254;
  private final static int APDU_HEADER_LENGTH =5;
  private final static int APDU_DATA_LENTH =
             MAX_APDU_LENGTH- APDU_HEADER_LENGTH;
  private ≪swtReaderName
    ()≫[] reader = new ≪swtReaderName
      ()≫[≪this.getModel()
        .getTerminal2CardPorts().size≫];
  public void initReader(SWTReader reader, int port) {
   this.reader[port] = reader;
  }
  public byte[] encode(≪codeableName
    ()≫ c)
      throws ≪terminalExceptionName()≫{
   byte[] encMsg = new byte[LengthConstants
     .MAX_ENCODING_LENGTH_MESSAGES];
   short encMsgLen = ≪codingName()≫
     .getInstance().encode(c, encMsg);
    return ByteArray.subarray(encMsg, 0, encMsgLen);
  }
   public byte[] transmit(byte[] apdu, int port)
     throws TerminalException, SWTReaderException {
    byte[] received=null;
    if(apdu.length <= MAX_APDU_LENGTH){
     apdu[4] = (byte)(apdu.length-5);
     received=reader[port].transmit(apdu);
     if(received.length==2)
      return received;
     received=ByteArray.subarray(received, 0,received
       .length-2);
    }
    else
    {
     byte[] splittedApdu;
     byte[] apduHeader=ByteArray
       .subarray(apdu, 0, APDU_HEADER_LENGTH);
     apdu=ByteArray
       .subarray(apdu, APDU_HEADER_LENGTH);
```

```
int splitCount=apdu.length/APDU_DATA_LENTH;
if(apdu.length % APDU_DATA_LENTH != 0)
 splitCount++;
for(int i=0; i<splitCount; i++){
 if(i==splitCount-1){
  splittedApdu=ByteArray
    .subarray(apdu, i*APDU_DATA_LENTH);
  apduHeader[2]= (byte)0;
  apduHeader[4]=(byte)splittedApdu.length;
  splittedApdu= ByteArray
    .append(apduHeader, splittedApdu);
  received = reader[port].transmit(splittedApdu);
  if(received.length==2)
   return received;
  received=ByteArray
    .subarray(received, 0,received.length-2);
 }
 else
 {
  splittedApdu=ByteArray
    .subarray
      (apdu, i*APDU_DATA_LENTH,APDU_DATA_LENTH);
  apduHeader[2]= (byte)1;
  apduHeader[4]=(byte)splittedApdu.length;
  splittedApdu = ByteArray
    .append(apduHeader, splittedApdu);
  received = reader[port].transmit(splittedApdu);
  if( !(received[0]==(byte)0x90 && received[1]==
    (byte)0x00) ){
   throw new TerminalException
     ("Bad answer from card: part of a long APDU was not
               answered with OK (0x9000).");
  }
 }
}
for(int i=0;i<MAX_SPLIT_LENGTH;i++){
 if(received[0]==0){
  received=ByteArray.subarray(received, 1);
  return received;
 }
 else if(received[0]==1){
  received=ByteArray.subarray(received, 1);
  byte[] getMore= new byte[]{0, 6, 0, 0, 0, 1, 1};
  byte[] temp_received = reader[port]
    .transmit(getMore);
  temp_received=ByteArray
    .subarray(temp_received, 0,temp_received
      .length-2);
  received=ByteArray
    .append(temp_received[0], received);
  temp_received= ByteArray
    .subarray(temp_received, 1);
  received=ByteArray
    .append(received, temp_received);
```

```
      }
      else
       throw new TerminalException
         ("Bad answer from card: answer to a long APDU
           must begin with 0 or 1.");
     }
    return received;
    }
≪ENDDEFINE≫

≪DEFINE generateAttributesAndMethodsForGates
  FOR Class≫
 public byte[] prevMsg;
 public ≪this.getModel().getUserMessage()
   .name≫ prevUMsg;
 public final ≪gateName
   ()≫ dummyGate = new ≪gateName()≫(){
 @Override
 public void callConstraints() {
 }
 @Override
 public byte[] transmit(≪this
   .name≫ terminal, byte[] apdu, int port)
     throws ≪terminalExceptionName()≫,
   SWTReaderException {
  return terminal.transmit(apdu, port);
 }
 @Override
 public void finishProtocol(≪this
   .name≫ terminal){
 }
 };
 public ≪gateName()≫ currentGate = dummyGate;
 public void reset≪gateName()≫(){
  currentGate = dummyGate;
 }
 public void set≪gateName()≫(≪gateName
   ()≫ gate){
  currentGate = gate;
 }
≪ENDDEFINE≫

≪DEFINE generateAttributesAndMethodsForUserConnection
  FOR Class≫
 ≪IF this.getModel().hasUserMessage()-≫
 private ≪userinterfaceName()≫ userinterface;
 ≪ENDIF-≫
 ≪IF this.getModel().hasUserMessage()-≫
 public void setUserinterface(Userinterface u){
  this.userinterface = u;
 }
 ≪ENDIF-≫
≪ENDDEFINE≫
```

```
≪DEFINE generateGate FOR Class≫
≪debug("Generating class Gate")≫
 ≪FILE this.getModel().getDirectoryPrefix()
  +gateName() + ".java"−≫
  package ≪this.getModel()
    .packagenameTerminal()−≫;
  import de.uniaugsburg.swt.javacard.terminalinterface
    .SWTReaderException;
  public interface ≪gateName()≫ {
  public void callConstraints();
  public byte[] transmit(≪this
    .name≫ terminal, byte[] apdu, int port)
  throws TerminalException, SWTReaderException;
  public void finishProtocol(≪this
    .name≫ terminal);
  }
 ≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateUserinterface FOR Model≫
≪debug("Generating interface "
  +userinterfaceName())−≫
 ≪FILE this.getModel().getDirectoryPrefix()
  +userinterfaceName()+".java"−≫
  package ≪this.getModel()
    .packagenameTerminal()−≫;
  public interface ≪userinterfaceName()−≫ {
   abstract void send(≪this.getModel()
     .getUserMessage().name≫ umsg);
  }
 ≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateUserMessage FOR Class−≫
≪FILE this.getModel().getDirectoryPrefix()
  +filename()−≫
  package ≪this.getModel()
    .packagenameTerminal()−≫;
  public class ≪name−≫≪IF this.general
    .size > 0−≫ extends ≪this.general.first()
      .name≫≪ENDIF−≫ {
   ≪EXPAND psm_templates_shared::classesShared
     ::Constructor FOR this≫
   ≪EXPAND psm_templates_shared::classesShared
     ::AttrEntry FOREACH this
       .getAttributesInCorrectOrder()≫
   ≪EXPAND psm_templates_shared::classesShared
     ::Getter FOREACH this.getAllAttributes()≫
  }
 ≪ENDFILE≫
≪ENDDEFINE≫
```

3.2.2 Generating the Coding (i.e the (de)serialisation) for the terminals

```
«IMPORT uml»
«EXTENSION psm_templates_shared::translations»
«EXTENSION psm_templates_shared::getter»
«EXTENSION psm_templates_shared::tests»
«EXTENSION psm_templates_shared::names»
«EXTENSION psm_templates_shared::parse»

«DEFINE main FOR uml::Model»
«logging("startcoding")»
 «FOREACH getTerminals().reject(e|e
   .isAbstract()) AS terminal»
  «setDirectoryPrefix(terminal)»
  «FILE getDirectoryPrefix()+codingName()
    + ".java"»
   package «this.getModel()
     .packagenameTerminal()-»;
   «LET packagedElement.typeSelect(Package)
     AS packages»
    «LET this.getCodeableClasses(terminal)
      .reject(c|c.isAbstract()) AS classes»
     «EXPAND psm_templates_shared::codingShared
       :: generateCodingHeader»
     «EXPAND psm_templates_shared::codingShared
       ::generateCodingMain(this.getModel())
         FOR classes»
     «EXPAND psm_templates_shared::codingShared
       :: generateCodingForClass FOREACH classes»
     «EXPAND psm_templates_shared::codingShared
       ::generateCodingFooter»
    «ENDLET»
   «ENDLET»
  «ENDFILE»
 «ENDFOREACH»
«ENDDEFINE»
```

3.2.3 Generating the class SimpleComm as well as other classes for the terminals

```
«IMPORT uml»
«EXTENSION psm_templates_shared::getter»
«EXTENSION psm_templates_shared::names»
«EXTENSION psm_templates_shared::tests»
«EXTENSION psm_templates_shared::parse»

«DEFINE main FOR uml::Model»
«FOREACH getTerminals().reject(e|e.isAbstract())
  AS terminal»
 «setDirectoryPrefix(terminal)»
 «EXPAND psm_templates_shared::simplecommShared
   ::generateSimplecommClasses FOR terminal»
«ENDFOREACH»
«ENDDEFINE»
```

3.3 Transformations that are used for both platforms (Smart Card and Terminal)

3.3.1 Transformations of a class diagram

```
«IMPORT uml»
«EXTENSION psm_templates_shared::translations»
«EXTENSION psm_templates_shared::parse»
«EXTENSION psm_templates_shared::getter»
«EXTENSION psm_templates_shared::tests»
«EXTENSION psm_templates_shared::names»

«DEFINE generateStubsGenerator FOR Class»
 «FILE this.getModel().getDirectoryPrefix()
   +stubsGeneratorName()+".java"-»
 package «this.getModel().packagename()-»;
 import java.io.*;
 import java.lang.reflect.Method;
 import java.util.*;
 import javax.net.ssl.SSLSession;
 «FOREACH getDirectlyUsedServers(this).reject(e|e
   .isAbstract()) AS s»
  import generated.«((String)GLOBALVAR package)
    .split("\\.").get(1)».server.«s.name
      .toFirstLower()».«s.name»;
 «IF s.isStateful()-»
  import generated.«((String)GLOBALVAR package)
    .split("\\.").get(1)».server.«s.name
      .toFirstLower()».«s
        .name»StatefulManager;
 «ENDIF-»
 «ENDFOREACH»
 public class «stubsGeneratorName()» {
  public static void main(String[] s)  {
   generateAllStubs();
  }
  public static void generateAllStubs(){
   «FOREACH getDirectlyUsedServers(this) AS s-»
   try{
    «IF s.isStateful()-»
     «LET s.name+"StatefulManager"
       AS serviceName»
     «LET (this.getModel().getServerNames()
       .indexOf(serviceName)+8000).toString()
         AS port-»
      «serviceName» «serviceName
        .toFirstLower()»=new «serviceName»
          ();
      «serviceName.toFirstLower()»
        .startService();
      generateStubs
        ("http://localhost:«port»/«serviceName»Service?wsdl"
          , "«serviceName
```

```
            .toFirstLower()≫", false);
     ≪serviceName.toFirstLower()≫
       .stopService();
    ≪ENDLET≫
    ≪ENDLET-≫
   ≪ENDIF-≫
   ≪LET (this.getModel().getServerNames()
     .indexOf(s.name)+8000).toString() AS port-≫
   ≪LET getIncommingAssociations(s.getModel(),s
     .name).first() AS serverSecurityChannel≫
   ≪LET serverSecurityChannel
     .hasStereotype("SecureMDD::TLS") AS TLS≫
   ≪s.name≫ ≪s.name
     .toFirstLower()≫=new ≪s.name≫();
   ≪s.name.toFirstLower()≫.startService();
   ≪IF TLS≫
    generateStubs
      ("https://localhost:≪port≫/≪s
        .name≫Service?wsdl", "≪s.name
          .toFirstLower()≫",≪TLS≫);
   ≪ELSE≫
    generateStubs
      ("http://localhost:≪port≫/≪s
        .name≫Service?wsdl", "≪s.name
          .toFirstLower()≫",≪TLS≫);
   ≪ENDIF≫
   ≪s.name.toFirstLower()≫.stopService();
   ≪ENDLET≫
   ≪ENDLET≫
   ≪ENDLET≫
 }catch(Exception ex){ex.printStackTrace();}
 ≪ENDFOREACH≫
}
public static void generateStubs
  (String serviceWSDLAddress, String serverName
    , Boolean TLS){
String modelName="≪((String)GLOBALVAR package)
  .split("\\.").get(1)≫";
String terminalName="≪this.name
  .toFirstLower()≫";
Boolean isTerminalServer=≪this.isServer()≫;
System.out.println("generateStubs in: "+modelName
  +"/"+terminalName+"/"+serverName);
String terminalServerPackage;
if(isTerminalServer)
 terminalServerPackage="server";
else
 terminalServerPackage="terminal";
String terminalPath="../Generated/src/generated/"
  +modelName+"/"+terminalServerPackage+"/"
    +terminalName;
String stubsPath="../Generated/src/generated/"
  +modelName+"/"+terminalServerPackage+"/"
    +terminalName+"/"+serverName+"Stubs";
```

```
String serverClassName=serverName.substring(0, 1)
  .toUpperCase()
    +serverName.substring(1,serverName.length());
String serverPackageName;
if (serverName.endsWith("StatefulManager"))
 serverPackageName=serverName
   .replaceAll("StatefulManager", "");
else
 serverPackageName=serverName;
try{
 System.out.println("compile server "+modelName
   +" package");
 System.out.println("startGenerateStubsInJava");
 if(TLS){
  System.setProperty("javax.net.ssl.trustStore",".
    ./Generated/src/META-INF/client-truststore.jks");
  System.setProperty("javax.net.ssl
    .trustStorePassword","changeit");
  System.setProperty("javax.net.debug","all");
  System.setProperty("javax.net.ssl.keyStore",".
    ./Generated/src/META-INF/client-keystore.jks");
  System.setProperty("javax.net.ssl
    .keyStorePassword","changeit");
  javax.net.ssl.HttpsURLConnection
    .setDefaultHostnameVerifier(
   new javax.net.ssl.HostnameVerifier(){
    @Override
    public boolean verify
      (String hostname, SSLSession arg1) {
     if (hostname.equals("localhost")) {
          return true;
        }
     return false;
    }
   }
  );
 }
 String[] a={"-keep","-d", ".
   ./Generated/src","-p","generated."+modelName+"."
     +terminalServerPackage+"."+terminalName+"."
       +serverName+"Stubs", serviceWSDLAddress};
 try {
  com.sun.tools.ws.WsImport.doMain(a);
 } catch (Throwable e) {
  e.printStackTrace();
 }
 System.out.println("delete Messages");
 List<File> stubFiles=getAllFilesWithSubDir(new File
   (stubsPath));
 List<File> terminalFiles= getAllFilesWithoutSubDir
   (new File(terminalPath));
 for(int i=0; i<stubFiles.size();i++){
  for(int j=0; j<terminalFiles.size();j++){
   if(terminalFiles.get(j).getName()
```

```
      .equals(stubFiles.get(i)
        .getName()) && !terminalFiles.get(j)
          .getName().equals("≪this.name≫
            .java") && !terminalFiles.get(j)
              .getName().equals("≪this
                .name≫StatefulManager
                  .java") && !terminalFiles.get(j)
                    .getName().endsWith("Exception
                      .java")){
     System.out.println(stubFiles.get(i).getName());
     stubFiles.get(i).delete();
    }
   }
  }
  System.out.println("add Import Terminal in Stubs");
  stubFiles= getAllFilesWithSubDir(new File
    (stubsPath));
  for(int i=0;i<stubFiles.size();i++)
   addImport(stubFiles.get(i),"import generated."
     +modelName+"."+terminalServerPackage+"."
       +terminalName+".*;");
 }catch(Exception ex){
  System.out.println("exception "+ex.getMessage());
 }
}
public static List<File> getAllFilesWithSubDir
  ( File path ) {
       List<File> ret = new ArrayList<File>();
       File[] files = path.listFiles();
       for ( File f : files ) {
           if ( f.isDirectory() ) {
               ret
                 .addAll( getAllFilesWithSubDir
                   ( f ) );
           } else {
               if ( f.getName().endsWith(".java") ) {
                   ret.add( f );
               }
           }
       }
       return ret;
 }
 public static List<File> getAllFilesWithoutSubDir
   ( File path ) {
        List<File> ret = new ArrayList<File>();
        File[] files = path.listFiles();
        for ( File f : files ) {
               if ( f.getName().endsWith(".java") ) {
                   ret.add( f );
               }
        }
        return ret;
 }
 public static void addImport
```

```
    (File file, String sImport) throws Exception{
   BufferedReader reader = new BufferedReader
     (new FileReader(file.getPath()));
   Vector<String> list= new Vector<String>();
   String s;
   while((s=reader.readLine())!=null){
    list.add(s);
    if(s.contains("package"))
     list.add(sImport);
   }
   reader.close();
   BufferedWriter writer = new BufferedWriter
     (new FileWriter(file.getPath()));
   for(int i=0;i<list.size();i++){
   writer.write(list.get(i));
    writer.write("\n");
   }
   writer.close();
  }
 }
 «ENDFILE-»
«ENDDEFINE»

«DEFINE generateMessageWrapper FOR Model»
 «FILE this.getModel().getDirectoryPrefix()
   +messageWrapperName()+".java"-»
 package «this.getModel().packagename()-»;
 import javax.xml.bind.annotation.*;
 @XmlAccessorType(XmlAccessType.FIELD)
 public class «messageWrapperName()» {
  @XmlElementRefs(
      {
   «LET this.getClassDiagramClasses().select(e|e
     .general.exists(e|e
       .hasStereotype("SecureMDD::Message")))
         AS messages-»
   «IF !messages.isEmpty-»
    @XmlElementRef( type = «messages.first()
      .name».class )
   «ENDIF-»
  «FOREACH messages.withoutFirst() AS e-»
   ,@XmlElementRef( type = «e.name».class )
  «ENDFOREACH-»
  «ENDLET-»
      } )
  private Message msg;
     public «messageWrapperName()»(){
     }
  public «messageWrapperName()»(Message msg){
   this.msg=msg;
  }
  public Message getMsg() {
   return msg;
  }
```

```
  public void setMsg(Message msg) {
   this.msg = msg;
  }
 }
 «ENDFILE-»
«ENDDEFINE»

«DEFINE generateException FOR Model»
 «FILE this.getModel().getDirectoryPrefix()
   +exceptionName()+".java"-»
 package «this.getModel().packagename()-»;
 public class «exceptionName()» extends java
   .lang.Exception {
  public «exceptionName()»() {
   super();
  }
  public «exceptionName()»(String s) {
   super(s);
  }
 }
 «ENDFILE-»
«ENDDEFINE»

«DEFINE generateManualClasses FOR List[Class]»
«FOREACH this AS c-»
«logging("Generating class "
  +c.name.toString())-»
«FILE c.filename() MANUAL-»
package «((String)GLOBALVAR package_manual)»;
import «c.getModel().packagename()».*;
public «IF c
  .isAbstract
    ()»abstract «ENDIF»class «c
      .name-»
«IF !c.getGeneralClasses()
  .isEmpty»extends «c.getGeneralClasses().name
    .first()» «ENDIF-»
«IF !c.getImplements()
  .isEmpty»implements «FOREACH c
    .getImplements()
      AS impl SEPARATOR "
        , "-»«impl-»«ENDFOREACH-»«ENDIF-»{
«FOREACH c.getAttributesInCorrectOrder() AS e»
 «e.getAttributeImplCode()»
«ENDFOREACH»
«EXPAND Constructor FOR c-»
«IF targetServer()||targetTerminal()»
 «EXPAND Getter FOREACH c.ownedAttribute-»
 «EXPAND Setter FOREACH c.ownedAttribute-»
«ENDIF-»
«IF c.implementsInterface("Codeable")-»
 «EXPAND makeCodeable FOR c-»
«ENDIF-»
«EXPAND generateEquals FOR c»
```

```
«EXPAND generateCopy FOR c»
«EXPAND generateMethods FOR c-»
}
«ENDFILE»
«ENDFOREACH»
«ENDDEFINE»

«DEFINE generateManualDebugClass
  (List[Operation] ops) FOR Class»
«logging("Generating ManualDebugClass "
  +this.name.toString())-»
«FILE this.manualFileName() MANUAL-»
package «this.getModel().packagenameManual()-»;
import «this.getModel().packagename()».*;
public class Manual_«this.name-»{
 «FOREACH ops AS op»
 «LET op.ownedParameter.reject(e|op.returnResult()
    .contains(e)) AS params»
  public static «op.returnResult().first()
    .translateReturnType()» «op
      .name»(«FOREACH params
        AS p SEPARATOR ", "»«p.translateType()
          .translatePrimitiveType()» «p
            .name»«ENDFOREACH» «IF !params
              .isEmpty» ,«ENDIF» «this
                .name» c){
   return «op.returnResult().first()
     .getEmptyDefaultReturnType()»;
  }
 «ENDLET»
 «ENDFOREACH»
}
«ENDFILE»
«ENDDEFINE»

«DEFINE generateMethods FOR Class»
«IF targetCard()»
«logging("Calling generateMethods with class "
  + this.name.toString() + ", predefined methods: "
    + this.getCustomMethods().toString())»
«ELSE»
«logging("Calling generateMethods with class "
  + this.name.toString())»
«ENDIF»
«FOREACH this.ownedOperation.select(e| e.name
  .toString().toFirstLower()!= this.name.toString()
    .toFirstLower()).reject(b|this.getCustomMethods()
      .exists( x|x.toString() == b.name
        .toString() )) AS op-»
«op.visibility.toString()»«IF op
  .isStaticOp()» static«ENDIF»«IF op
    .isAbstractOp
      () == true» abstract«ENDIF» «op
        .returnResult().first()
```

```
            .translateReturnType()» «op
              .name»(«FOREACH op.ownedParameter
                .reject(e|op.returnResult()
                  .contains(e))
                    AS p SEPARATOR ", "»«p
                      .translateType()
                        .translatePrimitiveType
                          ()» «p
                            .name»«ENDFOREACH»){
 «IF op.returnResult().first() != null && ! op
   .returnResult().first().translateType()
     .translatePrimitiveType()
       .isPrimitiveTranslatedType
         ()-»return null;«ENDIF-»
 }
«ENDFOREACH-»
«ENDDEFINE»

«DEFINE generateManualMethods FOR Class»
«IF this.hasManualMethods()»
 «logging
   ("Calling generateManualMethods with class "
     + this.name.toString())»
 «LET this.ownedOperation.union(this
   .getGeneralClass().ownedOperation).select(e| e
     .hasStereotype("SecureMDD::Manual")) AS ops»
 «IF isDebugMode()»
  «EXPAND generateManualDebugClass(ops
    .toList()) FOR this»
 «ENDIF»
 «PROTECT CSTART '' ID (this.getModel()
   .packagename()+"."+this.name)»
 «FOREACH ops AS op-»
 «op.visibility.toString()»«IF op
   .isStaticOp()» static«ENDIF»«IF op
     .isAbstractOp
       () == true» abstract«ENDIF» «op
         .returnResult().first()
           .translateReturnType()» «op
             .name»(«FOREACH op.ownedParameter
               .reject(e|op.returnResult()
                 .contains(e))
                   AS p SEPARATOR ", "»«p
                     .translateType()
                       .translatePrimitiveType
                         ()» «p
                           .name»«ENDFOREACH»){
 «IF isDebugMode()»
  «LET op.ownedParameter.reject(e|op
    .returnResult().contains(e)) AS params»
  «IF op.returnResult().first() != null -»
   return «this.manualClassName()».«op
     .name»(«FOREACH params
       AS p SEPARATOR ", "»«p
```

```
          .name≫«ENDFOREACH» «IF !params
            .isEmpty», «ENDIF» this);
  «ELSE»
   «this.manualClassName()».«op
     .name»(«FOREACH params
       AS p SEPARATOR ", "»«p
         .name»«ENDFOREACH»«IF !params
           .isEmpty», «ENDIF» this);
  «ENDIF»
  «ENDLET»
 «ELSE»
  return «op.returnResult().first()
    .getEmptyDefaultReturnType()»;
 «ENDIF»
 }
 «ENDFOREACH-»
 «ENDPROTECT»
 «ENDLET»
«ENDIF»
«ENDDEFINE»

«DEFINE generatePorts FOR Model»
 «FILE this.getModel().getDirectoryPrefix()
   +deploymentPropertiesName()+".java"-»
  package «this.getModel().packagename()-»;
  public class «deploymentPropertiesName()-» {
  «LET getModel()
    .getClass(deploymentPropertiesName())
      .ownedAttribute AS properties-»
   «FOREACH properties AS prop-»
    «prop.getAttributeImplCode()»
   «ENDFOREACH»
  «ENDLET»
  }
 «ENDFILE»
«ENDDEFINE»

«DEFINE generateLenghtConstants FOR Model»
«logging("Generating constants "
  +lengthConstantsName())-»
 «FILE this.getModel().getDirectoryPrefix()
   +lengthConstantsName() + ".java"»
 package «this.getModel().packagename()-»;
 public class «lengthConstantsName()» {
  «FOREACH getClass(this,lengthConstantsName())
    .ownedAttribute AS a»
   «a.getAttributeImplCode()»
  «ENDFOREACH»
 }
 «ENDFILE»
«ENDDEFINE»

«DEFINE generateClass FOR Class»
«logging("Generating class "
```

```
 +this.name.toString())-≫
≪FILE this.getModel().getDirectoryPrefix()
  +filename()-≫
 package ≪this.getModel().packagename()-≫;
  ≪IF !targetCard() &&this.general.exists(e|e
     .hasStereotype("SecureMDD::Message"))≫
 import javax.xml.bind.annotation.XmlRootElement;
 @XmlRootElement
 ≪ENDIF≫
 ≪IF this
    .hasManualMethods() && isDebugMode()≫
 import ≪this.getModel().packagenameManual()≫
    .*;
 ≪ENDIF≫
 public ≪IF this
    .isAbstract
      ()≫abstract ≪ENDIF≫class ≪name-≫
 ≪IF !this.getGeneralClasses()
    .isEmpty≫extends ≪this.getGeneralClasses()
      .name.first()≫ ≪ENDIF-≫
 ≪IF !this.getImplements()
    .isEmpty≫implements ≪FOREACH this
      .getImplements()
        AS impl SEPARATOR "
          , "-≫≪impl-≫≪ENDFOREACH-≫≪ENDIF-≫{
  ≪FOREACH this.instanceAttributes() AS e≫
   ≪e.getAttributeImplCode()≫
  ≪ENDFOREACH≫
  ≪EXPAND Constructor FOR this-≫
  ≪IF targetServer() || targetTerminal()-≫
   ≪EXPAND Getter FOREACH this
     .ownedAttribute- ≫
   ≪EXPAND Setter FOREACH this
     .ownedAttribute-≫
  ≪ENDIF-≫
  ≪IF this.implementsInterface("Codeable")-≫
   ≪EXPAND makeCodeable FOR this-≫
  ≪ENDIF-≫
  ≪EXPAND generateEquals FOR this≫
  ≪EXPAND generateCopy FOR this≫
  ≪IF !targetCard()≫
  ≪EXPAND generateCopyNewObject FOR this≫
  ≪EXPAND generateManualMethods FOR this≫
  ≪ELSEIF targetCard()&&this.hasMethods()≫
  ≪PROTECT CSTART '' ID (this.getModel()
    .packagenameCard()+"."+this.name)≫
  ≪EXPAND generateManualMethods FOR this≫
  ≪ENDPROTECT≫
  ≪ENDIF≫
  ≪IF targetCard()≫
   ≪IF this
     .hasStereotype("SecureMDD::Certificate")-≫
    ≪EXPAND generateCertificateMethodsforCard
      FOR this-≫
```

```
    «ENDIF-»
   «ELSEIF targetServer() || targetTerminal()»
    «IF this
      .hasStereotype("SecureMDD::Certificate")-»
     «EXPAND generateCertificateMethodsforTerminalAndServer
      FOR this-»
    «ENDIF-»
   «ENDIF»
  }
 «ENDFILE»
«ENDDEFINE»

«DEFINE generateCertificateMethodsforTerminalAndServer
  FOR Class»
public boolean verifyCertificate
  (PublicKey pubkey) throws «exceptionName()» {
  return SignedData.verify(pubkey, «this
    .getAttributesInCorrectOrder().select(e|e.type
      .name=="SignedData").first()
        .name», «this
          .getAttributesInCorrectOrder().select(e|e
            .type.name!="SignedData").first()
              .name»);
 }
public static «this
  .name» generateCertificate
    (PrivateKey issuerPrivateKey, «FOREACH this
      .getCertificateDataClass()
        .getAttributesInCorrectOrder()
          AS att SEPARATOR ","»«att
            .translateType()» «att
              .name»«ENDFOREACH»)
                throws «exceptionName()» {
   «this.getCertificateDataClass()
     .name» dataclass = new «this
       .getCertificateDataClass()
         .name»(«FOREACH this
           .getCertificateDataClass()
             .getAttributesInCorrectOrder()
               AS att SEPARATOR ", "»«att
                 .name»«ENDFOREACH»);
   SignedData s = SignedData
     .sign(issuerPrivateKey, dataclass);
   «IF this.getAttributesFromConstructor().first()
     .type.name == "SignedData"»
   return new «this.name»(s,dataclass);
   «ELSE»
   return new «this.name»(dataclass,s);
   «ENDIF»
 }
«ENDDEFINE»

«DEFINE generateCertificateMethodsforCard
  FOR Class»
```

```
public boolean verifyCertificate(PublicKey pubkey) {
  return SignedData.verify(pubkey, ≪this
    .getAttributesInCorrectOrder().select(e|e.type
      .name=="SignedData").first()
        .name≫, ≪this
          .getAttributesInCorrectOrder().select(e|e
            .type.name!="SignedData").first()
              .name≫);
 }
public static ≪this
  .name≫ generateCertificate
    (PrivateKey issuerPrivateKey, ≪FOREACH this
      .getCertificateDataClass()
        .getAttributesInCorrectOrder()
          AS att SEPARATOR ","≫≪att
            .translateType()≫ ≪att
              .name≫≪ENDFOREACH≫) {
   ≪this.getCertificateDataClass()
     .name≫ dataclass = Store.new≪this
       .getCertificateDataClass()
         .name≫(≪FOREACH this
           .getCertificateDataClass()
             .getAttributesInCorrectOrder()
               AS att SEPARATOR ", "≫≪att
                 .name≫≪ENDFOREACH≫);
   SignedData s = SignedData
     .sign(issuerPrivateKey, dataclass);
   ≪IF this.getAttributesFromConstructor().first()
     .type.name == "SignedData"≫
   return Store.new≪this.name≫(s,dataclass);
   ≪ELSE≫
   return Store.new≪this.name≫(dataclass,s);
   ≪ENDIF≫
 }
≪ENDDEFINE≫

≪DEFINE SetKeysAndReadIdentifiers FOR Class≫
 ≪IF ( (this.isServer() && !this
   .isStatefulManager()) || this
     .isTerminal() ) && !this.getDirectlyUsedServers()
       .isEmpty≫
  System.setProperty("javax.net.ssl.trustStore",".
    ./Generated/src/META-INF/client-truststore.jks");
  System.setProperty("javax.net.ssl
    .trustStorePassword","changeit");
  System.setProperty("javax.net.ssl.keyStore",".
    ./Generated/src/META-INF/client-keystore.jks");
  System.setProperty("javax.net.ssl
    .keyStorePassword","changeit");
  ≪IF this
    .callSeveralServiceInstancesWithSameComponentType
      ()≫
  readIdentifierToService();
  ≪ENDIF≫
```

```
 «ENDIF»
«ENDDEFINE»

«DEFINE Getter FOR Property-»
 public «this.translateType()» «this
   .getterName()»() {
  return «this.name»;
 }
«ENDDEFINE»

«DEFINE Setter FOR Property-»
 «IF this.isPrimitiveType()-»
  public void set«this.name.toString()
    .toFirstUpper()»(«this
      .translateType()» «this.name») {
   «IF this.translateType() == "String" && this
     .getModel().getCodeableClasses().contains(this
       .owner)-»
    if (!«codingName()».getInstance()
      .isValidCardString(«this.name»))
     System.err.println("Encoded string "
       + «this.name» + " is longer than "
         + «lengthConstantsName()»
           .LENGTHOFSTRING + " bytes.");
   «ENDIF-»
   this.«this.name» = «this.name»;
  }
  «IF targetTerminal() && this
    .translateType() == "String" && this.getModel()
      .getCodeableClasses().contains(this.owner)-»
   public void set«this.name.toString()
     .toFirstUpper()»(byte[] «this.name») {
    set«this.name.toString()
      .toFirstUpper()»(«codingName()»
        .getInstance().byteToString(«this
          .name»));
   }
  «ENDIF»
 «ELSEIF this.isPrimitiveTypeArray()-»
  public void set«this.name.toString()
    .toFirstUpper()»(«this
      .translateType()» «this
        .name»,short offset) {
   this.«this.name» = new «this
     .toCleanType()» [«this.name».length];
   Util.arrayCopy(«this.name-», offset, this
     .«this.name-», (short)0, (short) this
       .«this.name-».length);
  }
 «ELSE-»
  public void «this.setterName()»(«this
    .translateType()» «this.name») {
  this.«this.name»= «this.name»;
  «IF targetServer() && this.type
```

```
    .metaType != PrimitiveType && !this.getModel()
      .getClass(this.type.name).isListClass() && this
        .type.metaType != Enumeration-≫
   ≪FOREACH ((Class)this.type)
     .instanceAttributes() AS attr-≫
     this.≪this.name≫.set≪attr.name
       .toString().toFirstUpper()≫(≪this
         .name≫.get≪attr.name.toString()
           .toFirstUpper()≫());
   ≪ENDFOREACH-≫
  ≪ENDIF-≫
  }
 ≪ENDIF-≫
≪ENDDEFINE≫

≪DEFINE makeCodeable FOR Class≫
 public byte getCode() {
  return ≪codeName()≫.≪this.name
    .toString().toUpperCase()≫;
 }
≪ENDDEFINE≫

≪DEFINE generateEquals FOR Class≫
 public boolean equals(≪this.name≫ other) {
  ≪FOREACH attribute AS attr-≫
   ≪IF !targetCard()&& attr.isStringType()-≫
    if (! this.≪attr.name≫.equals(other
      .≪attr.name≫)) return false;
   ≪ELSEIF attr.isPrimitiveTypeArray()-≫
    if(! Arrays.equals(≪attr.name≫, other
      .≪attr.name≫)) return false;
   ≪ELSEIF attr.isPrimitiveType()-≫
    if(this.≪attr.name≫ != other.≪attr
      .name≫) return false;
   ≪ELSE-≫
    if (! this.≪attr.name≫.equals(other
      .≪attr.name≫)) return false;
   ≪ENDIF-≫
  ≪ENDFOREACH-≫
  return true;
 }
≪ENDDEFINE≫

≪DEFINE generateCopyNewObject FOR Class≫
≪IF !this.isAbstract()≫
 public ≪this.name
   .toString()≫ ≪copyName()≫() {
  ≪this.name
    .toString()≫ object = new ≪this.name
      .toString()≫();
  object.≪copyName()≫(this);
  return object;
 }
≪ENDIF≫
```

```
«ENDDEFINE»

«DEFINE generateCopy FOR Class»
«IF !this.isAbstract()»
 public void «copyName()»(«this.name
   .toString()» from) {
  «FOREACH attribute AS attr-»
   «IF attr.isPrimitiveType()-»
    this.«attr.name» = from.«attr
      .name»;
   «ELSEIF attr.isPrimitiveTypeArray()-»
    «IF !targetCard()»
    this.«attr.name» = from.«attr
      .name».clone();
    «ELSE»
    Arrays.copy(from.«attr.name», this
      .«attr.name»);
    «ENDIF»
   «ELSE-»
    this.«attr.name».«copyName()»(from
      .«attr.name»);
   «ENDIF-»
  «ENDFOREACH-»
 }
«ENDIF»
«ENDDEFINE»

«DEFINE generateDumpStubs FOR Node-»
 «FILE this.getModel().getDirectoryPrefix()+"/"
   +this.name.toFirstLower()+"Stubs/"+this.name
     +".java"-»
  package «this.getModel().packagename()»
    .«this.name.toFirstLower()»Stubs;
  import «this.getModel().packagename()».*;
  public class «this.name-»{
   public «messageWrapperName()» process
     («messageWrapperName()» m){return null;}
   public void closeService(){}
  }
 «ENDFILE»
 «FILE this.getModel().getDirectoryPrefix()+"/"
   +this.name.toFirstLower()+"Stubs/"+this.name
     +"Service.java"-»
  package «this.getModel().packagename()»
    .«this.name.toFirstLower()»Stubs;
  public class «this.name-»Service{
   public «this.name-»Service(){}
   public «this
     .name-»Service(Object o1, Object o2){}
   public «this
     .name-» getPort
       (Object o1, Object o2){return null;}
   public «this.name-» get«this
     .name-»Port(){return null;}
```

```
  }
 «ENDFILE»
«IF this.isStateful()»
 «FILE this.getModel().getDirectoryPrefix()+"/"
   +this.name.toFirstLower()+"StatefulManagerStubs/"
     +this.name+"StatefulManager.java"-»
  package «this.getModel().packagename()»
    .«this.name
      .toFirstLower()»StatefulManagerStubs;
  public class «this.name-»StatefulManager{
   public Object getReference(){return null;}
  }
 «ENDFILE»
 «FILE this.getModel().getDirectoryPrefix()+"/"
   +this.name.toFirstLower()+"StatefulManagerStubs/"
     +this.name+"StatefulManagerService.java"-»
  package «this.getModel().packagename()»
    .«this.name
      .toFirstLower()»StatefulManagerStubs;
  public class «this
    .name-»StatefulManagerService{
   public «this
     .name-»StatefulManagerService() {}
   public «this
     .name-»StatefulManagerService(java.net
       .URL url, javax.xml.namespace.QName qName) {}
   public «this
     .name-»StatefulManager get«this
       .name-»StatefulManagerPort(){return null;}
  }
 «ENDFILE»
«ENDIF»
«ENDDEFINE»

«DEFINE generateDumpServiceIdentifierFile
  FOR Class-»
«IF this
  .callSeveralServiceInstancesWithSameComponentType
    ()»
 «FILE this.getModel().getDirectoryPrefixClass()
   .name.toFirstLower()
     +"/IdentifierToService.json" MANUAL-»
 «ENDFILE»
«ENDIF»
«ENDDEFINE»

«DEFINE AttrEntry FOR Property-»
  public «IF ((Class)this.owner)
    .hasStaticAttributes
      ()-»static «ENDIF» «this
        .translateType()-» «this.name»;
«ENDDEFINE»

«DEFINE generateActivities FOR Class»
```

```
 «EXPAND printActivity(this) FOREACH this
   .getAllInitNodes()»
«ENDDEFINE»

«DEFINE printActivity(Class class)
  FOR ActivityNode-»
«debug("partClass: "
  + this.inPartition.first().getPartitionClass()
    + ", class: " + class.name.toString()
      + ", actNode: " + this.name.toString())-»
  «IF this.isInputParameterNode() || this
    .metaType==InitialNode -»
   «LET ((Activity)(this.owner)).ownedParameter
     AS params-»
   «LET params.select(e|e.direction
     .toString()=="return") AS resparams-»
   «LET params.select(e|e.direction
     .toString()=="in" || e.direction
       .toString()=="inout") AS inparams-»
     private «IF resparams
       .size==0»void«ELSE»«resparams
         .first().translateType()
           .translatePrimitiveType
             ()»«ENDIF-»
     «((Activity)(this.owner))
       .name»(«FOREACH inparams
         AS param SEPARATOR ", "-»«param
           .translateType()
             .translatePrimitiveType()» «param
               .name»«ENDFOREACH-»)
                 throws «exceptionName()»{
     «IF resparams.size != 0»«resparams
       .first().translateType()» «resparams
         .first().name»«IF resparams.first()
           .translateType()
             .isPrimitiveType
               ()»; «ELSE»=new «resparams
                 .first()
                   .translateType()»
                     (); «ENDIF»«ENDIF-»
     «LET ((Activity)(this.owner)).node
       AS innernodes-»
     «LET innernodes.typeSelect(ForkNode).outgoing
       .target AS start-»
     «IF start.size == 0-»
      «EXPAND printActivity(class)
        FOR this.outgoing.target.first()-»
     «ELSE-»
      «EXPAND printActivity(class)
        FOR start.first()-»
     «ENDIF-»
     }
   «ENDLET-»«ENDLET-»«ENDLET-»«ENDLET-»
   «ENDLET-»
```

```
≪ENDIF–≫
≪IF this.isReturnParameterNode()–≫
 ≪IF this.name.toString()
   .length>0 && ((ActivityParameterNode)this)
     .isStructuredActivityEnd()–≫
  return ≪this.name–≫;
 ≪ENDIF–≫
≪ENDIF–≫
≪IF (this.metaType==AcceptEventAction || this
  .hasStereotype
    ("SecureMDD::closeSession")) && class.getModel()
      .getAllClasses().select(e|this.getLabel()
        .toString().startsWith(e.name.toString()))
          .first().name.toString().length>0–≫
 ≪this.getLabel().toString()–≫
   {
   ≪IF !targetCard() && class.getModel()
     .getClassDiagramClasses().select(e|e
       .hasStereotype("SecureMDD::Service"))
         .size>0–≫
    synchronized≪IF isStateful(class)–≫
      (manager)≪ELSE–≫(this)≪ENDIF–≫
    {
  ≪EXPAND printActivity(class) FOREACH this
    .outgoing.target–≫
  }
 ≪ELSE–≫
  ≪EXPAND printActivity(class) FOREACH this
    .outgoing.target–≫
 ≪ENDIF–≫
 }
≪ENDIF–≫
≪IF this.metaType==DecisionNode–≫
 ≪IF ((DecisionNode)this).isMergeNode()–≫
  ≪EXPAND printActivity(class)
    FOR this.outgoing.first().target–≫
 ≪ELSE–≫
  if(≪this.outgoing.select(e|e.guard
    .stringValue().toString()!="else").first().guard
      .stringValue().toString()–≫) {
   ≪EXPAND printActivity(class) FOREACH this
     .outgoing.select(e|e.guard.stringValue()
       .toString()!="else").target–≫
  }
  else {
   ≪EXPAND printActivity(class) FOREACH this
     .outgoing.select(e|e.guard.stringValue()
       .toString()=="else").target–≫
  }
 ≪ENDIF–≫
≪ENDIF–≫
≪IF this
  .metaType==SendSignalAction || !targetCard
    () && this
```

```
        .hasStereotype("SecureMDD::openSession")-≫
    ≪this.getLabel().toString()≫
  ≪ENDIF-≫
  ≪IF this.metaType==CallBehaviorAction-≫
   ≪this.name.toString()≫
   ≪EXPAND printActivity(class) FOREACH this
     .outgoing.target-≫
  ≪ENDIF-≫
  ≪IF this.metaType==FlowFinalNode-≫
   ≪abortName()≫();
   ≪LET ((Activity)this.owner).isOperation()
     AS isInSubAction-≫
    ≪IF (isInSubAction)-≫
     ≪LET ((Activity)(this.owner)).ownedParameter
       .select(e|e.direction.toString()=="return")
         AS resparams-≫
     ≪IF resparams.size == 1-≫
      return ≪resparams.first()
        .getEmptyDefaultReturnType()≫;
     ≪ENDIF-≫
     ≪ENDLET-≫
    ≪ELSEIF (!isInSubAction && targetServer())≫
     return null;
    ≪ENDIF-≫
   ≪ENDLET-≫
  ≪ENDIF-≫
  ≪IF !targetCard()&&this
    .metaType==ActivityFinalNode-≫
  ≪ENDIF-≫
≪ENDDEFINE≫

≪DEFINE Constructor FOR Class≫
 ≪EXPAND InitConstructor FOR this≫
≪ENDDEFINE≫

≪DEFINE InitConstructor FOR Class≫
 ≪LET this.instanceOperations().select(e|e.name
   .toString()==this.name.toString()) AS consops-≫
  ≪FOREACH consops AS op≫
    public ≪op.name≫(≪FOREACH op
      .ownedParameter
        AS param SEPARATOR ","≫≪op.class
          .getPropertyForParam(param)
            .translateType()≫ ≪param
              .name≫≪ENDFOREACH≫) ≪IF this
                .isServer() || this
                  .isTerminal()≫
                    throws ≪exceptionName
                      ()≫ ≪ENDIF≫{
    ≪IF !targetCard()≫
    ≪ENDIF≫
     ≪FOREACH op.ownedParameter AS param-≫
      this.≪op.class.getPropertyForParam(param)
        .name≫=≪param.name≫;
```

```
    «ENDFOREACH-»
    «FOREACH getAllAttributesG().select(e|e
      .hasInitValue()).reject(e|op.ownedParameter
        .name.contains(e.name)) AS attr-»
     «attr.name»=«attr
       .initialValue()»;
    «ENDFOREACH-»
    «EXPAND SetKeysAndReadIdentifiers
      FOR this»
   }
   «IF targetTerminal()»
    «IF op.ownedParameter.exists(e|e
      .translateType() == "String")-»
     public «op.name»(«FOREACH op
       .ownedParameter AS param SEPARATOR ","»
      «IF op.class.getPropertyForParam(param)
        .translateType() == "String"-»
       byte[]
      «ELSE-»
       «op.class.getPropertyForParam(param)
         .translateType()»
      «ENDIF-»
       «param.name»«ENDFOREACH») {
      «FOREACH op.ownedParameter AS param-»
       «IF param.translateType() == "String"-»
        this.«op.class.getPropertyForParam(param)
          .name» = «codingName()»
            .getInstance().byteToString(«param
              .name»);
       «ELSE-»
        this.«op.class.getPropertyForParam(param)
          .name»=«param.name»;
       «ENDIF-»
      «ENDFOREACH-»
      «FOREACH getAllAttributesG().select(e|e
        .hasInitValue()).reject(e|op.ownedParameter
          .name.contains(e.name)) AS attr-»
       «attr.name»=«attr
         .initialValue()»;
      «ENDFOREACH-»
      «EXPAND SetKeysAndReadIdentifiers
        FOR this»
     }
    «ENDIF-»
   «ENDIF-»
  «ENDFOREACH-»
 «ENDLET»
«ENDDEFINE»

«DEFINE generateExceptionMehtod FOR Class»
 protected static void «abortName()»()
   throws «exceptionName()»{
  throw new «exceptionName()»
    ("«name-»");
```

```
 }
«ENDDEFINE»

«DEFINE generateAllAttributes FOR Class»
 «FOREACH this.getAllAttributesG() AS p»
  «p.getAttributeImplCode()»
 «ENDFOREACH»
«ENDDEFINE»

«DEFINE genrateSetTimeoutMethod FOR Class»
 «IF !this.getDirectlyUsedServers().isEmpty-»
 private static void setTimeout(BindingProvider b){
  b.getRequestContext().put("com.sun.xml.ws.request
    .timeout", new Integer(1000*60*10));
  b.getRequestContext().put("com.sun.xml.ws.connect
    .timeout", new Integer(1000*60*10));
 }
 «ENDIF-»
«ENDDEFINE»

«DEFINE generateStubsAttributes FOR Class»
 «IF callSeveralServiceInstancesWithSameComponentType
   ()»
  private java.util
    .Map<String, String[]> serviceIdentifiers= java
      .util.Collections.synchronizedMap(new java.util
        .TreeMap<String,String[]>());
 «ENDIF»
 «IF this.getDirectlyUsedServers().exists(e|e
   .isStateful())»
  private java.util
    .Map<String, Object> statefulServices=java.util
      .Collections.synchronizedMap(new java.util
        .TreeMap<String,Object>());
 «ENDIF»
«ENDDEFINE»

«DEFINE generateImportManuals FOR Class»
 «IF this.getModel().hasManualClasses()»
  import «((String)GLOBALVAR package_manual)»
    .*;
 «ENDIF»
«ENDDEFINE»

«DEFINE generateImportStubs FOR Class»
 «FOREACH this.getDirectlyUsedServers().reject(e|e
   .isAbstract()) AS s»
  import «((String)GLOBALVAR package)»
    .«this.name.toFirstLower()».«s.name
      .toFirstLower()»Stubs.*;
 «IF s.isStateful()-»
  import «((String)GLOBALVAR package)»
    .«this.name.toFirstLower()».«s.name
      .toFirstLower()»StatefulManagerStubs.*;
```

```
 «ENDIF-»
 «ENDFOREACH»
«ENDDEFINE»

«DEFINE generateMethodsForInvokingMultipleInstantiatedServices
  FOR Class»
«IF callSeveralServiceInstancesWithSameComponentType
  ()»
 private «IF targetServer()» «this
   .getModel()
     .getNameMessageSuperclass
       ()» «ELSE» void «ENDIF» sendMsg
         («this.getModel()
           .getNameMessageSuperclass
             ()» msg, String identifier)
               throws «exceptionName()»{
  «IF targetServer()»
   return sendMsg(msg, -1, identifier,false,false);
  «ELSE»
   sendMsg(msg, -1, identifier,false,false);
  «ENDIF»
 }
 private «IF targetServer()» «this
   .getModel()
     .getNameMessageSuperclass
       ()» «ELSE» void «ENDIF» sendMsg
         («this.getModel()
           .getNameMessageSuperclass
             ()» msg, String identifier
               , Boolean openSessionBeforeSend
                 , Boolean closeSessionAfterReceive)
                   throws «exceptionName()»{
  «IF targetServer()»
   return sendMsg
     (msg, -1, identifier, openSessionBeforeSend
       , closeSessionAfterReceive);
  «ELSE»
   sendMsg
     (msg, -1, identifier, openSessionBeforeSend
       , closeSessionAfterReceive);
  «ENDIF»
 }
 private «IF targetServer()» «this
   .getModel()
     .getNameMessageSuperclass
       ()» «ELSE» void «ENDIF» sendMsg
         («this.getModel()
           .getNameMessageSuperclass
             ()» msg, int port
               , String identifier)
                 throws «exceptionName()»{
  «IF targetServer()»
   return sendMsg(msg, port, identifier,false,false);
  «ELSE»
```

```
  sendMsg(msg, port, identifier,false,false);
 ≪ENDIF≫
}
private ≪IF targetServer()≫ ≪this
  .getModel()
    .getNameMessageSuperclass
      ()≫ ≪ELSE≫ void ≪ENDIF≫ sendMsg
        (≪this.getModel()
          .getNameMessageSuperclass
            ()≫ msg, int port, String identifier
              , Boolean openSessionBeforeSend
                , Boolean closeSessionAfterReceive)
                  throws ≪exceptionName()≫{
 ≪this.getModel()
   .getNameMessageSuperclass()≫ response=null;
 Object receiver = getReceiver
   (identifier,port, openSessionBeforeSend);
≪FOREACH this.getDirectlyUsedServers()
  .select(e|this
    .callSeveralInstancesFromThisComponent(e))
      AS server≫
 ≪IF server.name==this.name≫
  if(receiver instanceof ≪this.name≫){
   ≪this.name≫ s = (≪this
     .name≫)receiver;
   response = s.process(new MessageWrapper(msg))
     .getMsg();
  ≪IF targetServer()≫return processMessage
    (response);≪ENDIF≫
  ≪IF !targetServer()≫processMessage
    (response); return;≪ENDIF≫
  }
 ≪ELSE≫
  if(receiver instanceof ≪server.getModel()
    .packagename()≫.≪server.name
      .toFirstLower()≫Stubs.≪server
        .name≫){
   ≪this.getModel().packagename()≫
     .≪server.name.toFirstLower()≫Stubs
       .≪server.name≫ s = (≪this
         .getModel().packagename()≫.≪server
           .name.toFirstLower()≫Stubs.≪server
             .name≫)receiver;
   setTimeout((BindingProvider)s);
   try{
    response = s.process(new MessageWrapper(msg))
      .getMsg();
   }catch(java.lang
     .Exception e){≪abortName()≫();}
  ≪IF targetServer()≫
   ≪this.getModel()
     .getNameMessageSuperclass
       ()≫ responseMsg= processMessage(response);
   ≪IF server.isStateful()≫
```

```
    if(closeSessionAfterReceive){
     s.closeService();
    }
   ≪ENDIF≫
   return responseMsg;
  ≪ELSE≫
   processMessage(response);
   return;
  ≪ENDIF≫
  }
 ≪ENDIF≫
≪ENDFOREACH≫
 stop();
 ≪IF targetServer()≫
 return null;
 ≪ENDIF≫
}
private Object getReceiver
  (String identifier,int port
    , Boolean openSessionBeforeSend)
      throws ≪exceptionName()≫{
 try{
  ≪IF this.getDirectlyUsedServers().exists(e|e
    .name==this.name && this
      .callSeveralInstancesFromThisComponent(e))≫
  if(identifier.equals(this.≪this
    .getServiceIdentifier().name≫))
   return this;
  ≪ENDIF≫
  ≪LET this.getNode().getAllAttributes()
    .select(a|a.type
      .hasStereotype("SecureMDD::Service") && a.type
        .name.getNode(this.getModel())
          .isStateful() && this
            .callSeveralInstancesFromThisComponent(a
              .type.getNode())).type
                AS statefulServers≫
  ≪IF !statefulServers.isEmpty≫
  if(openSessionBeforeSend){
   ≪FOREACH statefulServers AS server≫
    if(getReceiverType(identifier)
      .equals("≪server.name≫")){
     ≪this.getModel().packagename()≫
       .≪server.name
         .toFirstLower()≫StatefulManagerStubs
           .≪server
             .name≫StatefulManager manager=
      new ≪server
        .name≫StatefulManagerService(new java
          .net
            .URL(getReceiverAddress
              (identifier)),new javax.xml.namespace
                .QName("≪server.name
                  .getNamespace
```

```
                    ("server")≫", "≪server
                      .name≫StatefulManagerService"))
                        .get≪server
                          .name≫StatefulManagerPort
                            ();
      setTimeout((BindingProvider)manager);
      ≪this.getModel().packagename()≫
        .≪server.name.toFirstLower()≫Stubs
          .≪server
            .name–≫ statefulService= new ≪server
              .name≫Service().getPort(manager
                .getReference(), ≪this.getModel()
                  .packagename()≫.≪server
                    .name.toFirstLower()≫Stubs
                      .≪server.name–≫.class);
      setTimeout((BindingProvider)statefulService);
      statefulServices.put(identifier
        + (port!=-1 ? " via "
          +port : ""),statefulService);
    }
  ≪ENDFOREACH≫
 }
 ≪ENDIF≫
 ≪ENDLET≫
 ≪FOREACH this.getNode().attribute.select(e|this
   .callSeveralInstancesFromThisComponent(e.type
     .getNode())).type AS s≫
 if(getReceiverType(identifier).equals("≪s
   .name≫"))
 ≪IF s.name.getNode(this.getModel())
   .isStateful()≫
  return statefulServices.get(identifier
   + (port!=-1 ? " via "+port : "") );
 ≪ELSE≫
  return new ≪s.name≫Service(new java.net
    .URL(getReceiverAddress(identifier)),new javax
      .xml.namespace.QName("≪s.name
        .getNamespace("server")≫", "≪s
          .name≫Service")).get≪s
            .name≫Port();
 ≪ENDIF≫
 ≪ENDFOREACH≫
}catch(Exception e){throw new ServiceException(); }
stop();
return null;
}
private String getReceiverAddress(String identifier){
 return this.serviceIdentifiers.get(identifier)[0];
}
private String getReceiverType(String identifier) {
 return this.serviceIdentifiers.get(identifier)[1];
}
private void readIdentifierToService()
  throws ≪exceptionName()≫ {
```

```
  com.google.gson.Gson gson = new com.google.gson
     .Gson();
  java.io.File file= new java.io.File(".
     ./Generated/src/≪this.getModel()
       .packagenameManual()
         .packageToPathName()≫/IdentifierToService
           .json");
  if(file.exists()){
   try{
    java.io.BufferedReader reader = new java.io
       .BufferedReader(new java.io.FileReader(file
         .getPath()));
    String s;
    StringBuffer sb=new StringBuffer();
    while ((s = reader.readLine()) != null)
     sb.append(s);
    java.lang.reflect.Type typeOfMap = new com.google
       .gson.reflect.TypeToken<java.util
         .Map<String, String[]>>() {}.getType();
        serviceIdentifiers = gson.fromJson(sb
          .toString(), typeOfMap);
   }catch(java.lang.Exception e){
    throw new ≪exceptionName()≫();}
  }
  else
   throw new ≪exceptionName()≫
     ("IdentifierToService.json does not exist");
 }
≪ENDIF≫
≪ENDDEFINE≫

≪DEFINE generateSendMsgMethods FOR Class≫
 private ≪IF targetServer()≫ ≪this
   .getModel()
     .getNameMessageSuperclass
       ()≫ ≪ELSE≫ void ≪ENDIF≫ sendMsg
         (≪this.getModel()
           .getNameMessageSuperclass()≫ msg){
 ≪IF targetServer()≫
  return msg;
 ≪ENDIF≫
 ≪IF targetTerminal()≫
  currentGate.callConstraints();
  userinterface.send((≪this.getModel()
    .getUserMessage().name≫)msg);
 ≪ENDIF≫
 }
 private ≪IF targetServer()≫ ≪this
   .getModel()
     .getNameMessageSuperclass
       ()≫ ≪ELSE≫ void ≪ENDIF≫ sendMsg
         (≪this.getModel()
           .getNameMessageSuperclass
             ()≫ msg, int port)
```

```
                throws «exceptionName()» {
 «IF targetServer()»
  return sendMsg(msg, port, false,false);
 «ELSE»
  sendMsg(msg, port, false,false);
 «ENDIF»
}
private «IF targetServer()» «this
  .getModel()
    .getNameMessageSuperclass
      ()» «ELSE» void «ENDIF» sendMsg
        («this.getModel()
          .getNameMessageSuperclass
            ()» msg, int port
              , Boolean openSessionBeforeSend
                , Boolean closeSessionAfterReceive)
                  throws «exceptionName()» {
 «IF targetTerminal()»
 currentGate.callConstraints();
 «ENDIF»
 «this.getModel()
   .getNameMessageSuperclass()» response=null;
 switch(port){
 «FOREACH this.getNode().attribute AS port-»
  «IF port.type.isServer() && !this
    .callSeveralInstancesFromThisComponent(port.type
      .getNode())-»
    case Ports.«port.name»:
    «IF port.type.name.getNode(this.getModel())
      .isStateful()-»
     if(openSessionBeforeSend){
      «LET ((Node)port.type) AS server»
       «server
         .name»StatefulManager manager«server
           .name-»= new «server
             .name»StatefulManagerService()
               .get«server
                 .name»StatefulManagerPort();
       setTimeout((BindingProvider)manager«server
         .name-»);
       «server
         .name-» statefulService«server
           .name-»=null;
       try{
        statefulService«server
          .name-»= new «server
            .name»Service()
              .getPort(manager«server.name-»
                .getReference(), «server
                  .name».class);
       }catch(java.lang.Exception e){
        «abortName()»();
       }
       setTimeout(
```

```
        (BindingProvider)statefulService≪server
          .name–≫);
       statefulServices.put(Integer
         .toString(port), statefulService≪server
           .name–≫);
      ≪ENDLET≫
     }
     try{
      response= ((≪port.type
        .name≫)statefulServices.get(Integer
          .toString(port)))
            .process(new ≪messageWrapperName
              ()≫(msg)).getMsg();
     }catch(java.lang.Exception e){
      ≪abortName()≫();
     }
     if(closeSessionAfterReceive){
      ((≪port.type.name≫)statefulServices
        .get(Integer.toString(port))).closeService();
     }
    ≪ELSE–≫
     ≪this.getModel().packagename()≫
       .≪port.type.name.toFirstLower()≫Stubs
         .≪port.type.name≫ ≪port.type
           .name.toFirstLower()≫ =new ≪port
             .type.name≫Service().get≪port
               .type.name≫Port();
     setTimeout((BindingProvider)≪port.type.name
       .toFirstLower()≫);
     try{
      response= ≪port.type.name
        .toFirstLower()≫
          .process(new ≪messageWrapperName()≫
            (msg)).getMsg();
     }catch(java.lang.Exception e){
      ≪abortName()≫();
     }
    ≪ENDIF–≫
    ≪IF targetServer
      ()≫ return ≪ENDIF–≫
     processMessage(response);
    ≪IF targetTerminal
      ()≫ break; ≪ENDIF≫
  ≪ELSEIF port.type.isSmartcard() –≫
    case Ports.≪port.name≫:
     response = sendCardMsg(msg,port);
     if(response!=null)
    ≪IF targetServer
      ()≫ return ≪ENDIF–≫
     processMessage(response);
    ≪IF targetTerminal
      ()≫ break; ≪ENDIF≫
  ≪ENDIF≫
≪ENDFOREACH–≫
```

```
   default :
    ≪abortName()≫();
    ≪IF targetServer()-≫
    return null;
    ≪ENDIF-≫
  }
 }
 ≪IF targetTerminal()≫
 private ≪this.getModel()
   .getNameMessageSuperclass()≫ sendCardMsg
     (≪this.getModel()
       .getNameMessageSuperclass
         ()≫ msg, int port)
           throws ≪exceptionName()≫ {
  byte[] encMsg = new byte[LengthConstants
    .MAX_ENCODING_LENGTH_MESSAGES];
  short encMsgLen = ≪codingName()≫
    .getInstance().encode((Codeable)msg, encMsg);
  encMsg = ByteArray.subarray(encMsg, 0, encMsgLen);
  byte[] apdu = ByteArray
    .append(new byte[] { 0, 2, 0, 0, 0}, encMsg);
  byte[] response;
  try {
   response = currentGate.transmit(this, apdu, port);
  } catch (SWTReaderException e) {
   ≪abortName()≫();
   return null;
  }
  if (response.length == 2) {
   if(!(response[0] == (byte)0x90 && response[1] ==
     (byte)0x00))
    throw new TerminalException("Cardexception: "
      + HandleResult.interpretSW(response) );
   return null;
  } else{
   prevMsg = response;
   Coding.getInstance().decodeInit();
   Codeable cResponse=Coding.getInstance()
     .decode(response);
   return (≪this.getModel()
     .getNameMessageSuperclass()≫) cResponse;
  }
 }
 ≪ENDIF≫
≪ENDDEFINE≫
```

3.3.2 Transformations that are used for generation of class Coding

```
≪IMPORT uml≫
≪EXTENSION psm_templates_shared::translations≫
≪EXTENSION psm_templates_shared::getter≫
≪EXTENSION psm_templates_shared::tests≫
≪EXTENSION psm_templates_shared::names≫
≪EXTENSION psm_templates_shared::parse≫
```

```
≪DEFINE generateCodingHeader FOR Package≫
≪IF targetServer()||targetTerminal()≫
import swt.util.ByteArray;
import java.nio.charset.Charset;≪ENDIF≫
≪IF targetCard()≫
import javacard.framework.ISO7816;
import javacard.framework.ISOException;
import javacard.framework.Util;≪ENDIF≫
public class ≪codingName()≫ {
 private short encoding_length;
 private byte[] encodeDestination;
 private static ≪codingName()≫ the_instance;
 ≪IF targetServer()||targetTerminal()≫
 private static Charset charset = Charset
   .forName("US-ASCII");≪ENDIF≫
 public static ≪codingName()≫ getInstance() {
  if
    (the_instance == null) the_instance = new ≪codingName
      ()≫();
  return the_instance;
 }
 public short getEncodingLength() {
  return encoding_length;
 }
 public void encodeInit(byte[] dest) {
  encodeDestination = dest;
  encoding_length = 0;
 }
 public void decodeInit() {
  encoding_length = 0;
 }
 ≪IF targetServer() ||targetTerminal()≫
≪ENDIF≫
  private byte decodeByte
    (byte[] in) ≪IF targetTerminal
      () || targetServer()≫
        throws ≪exceptionName
          ()≫≪ENDIF≫ {
  if (in[encoding_length++] != Code.BYTE){
  ≪IF targetTerminal() || targetServer
    ()≫ ≪abortName()≫();≪ENDIF≫
  ≪IF targetCard()≫
  ≪abortClassName()≫(); ≪ENDIF≫ }
  byte res = in[encoding_length++];
  return res;
 }
 private boolean decodeBoolean
   (byte[] in)≪IF targetTerminal() || targetServer
     ()≫
       throws ≪exceptionName()≫≪ENDIF≫{
  if (in[encoding_length++] != Code.BOOLEAN){
   ≪IF targetTerminal() || targetServer
     ()≫ ≪abortName()≫();≪ENDIF≫
  ≪IF targetCard()≫ ≪abortClassName()≫
```

```
    (); ≪ENDIF≫
   }
  boolean res = ( in[encoding_length+
    +] == 1 ? true : false );
  return res;
}
≪IF targetTerminal() || targetServer()≫
public ≪codeableName()≫ decode
   (byte[] in, short offset)
     throws ≪exceptionName()≫{
  encoding_length=offset;
  return decode(in);
}
public byte[] stringToByte(String s) {
  byte[] b = s.getBytes(charset);
  byte[] ret = new byte[≪lengthConstantsName
    ()≫.LENGTHOFSTRING];
  int len = (b
    .length < ≪lengthConstantsName()≫
      .LENGTHOFSTRING) ? b.length : LengthConstants
        .LENGTHOFSTRING;
  ByteArray.copy(b, 0, ret, 0, len);
  return ret;
}
public boolean isValidCardString(String s){
  return (s.getBytes(charset)
    .length <= ≪lengthConstantsName()≫
      .LENGTHOFSTRING);
}
public void encodeString(String i) {
  byte[] o = stringToByte(i);
  encodeDestination[encoding_length+
    +] = ≪codeName()≫.BYTEARRAY;
  ByteArray
    .setShort(encodeDestination, encoding_length,
      (short)(o.length));
  encoding_length += 2;
  ByteArray
    .copy(o, 0, encodeDestination, encoding_length, o
      .length);
  encoding_length += o.length;
}
private void compSizeString(String i) {
  byte[] o = stringToByte(i);
  encoding_length += 3;
  encoding_length += o.length;
}
≪ENDIF≫
  public short encode
    (Codeable o, byte[] dest) ≪IF targetServer
      ()||targetTerminal()≫
        throws ≪exceptionName
          ()≫ ≪ENDIF≫ {
  encodeInit(dest);
```

```
 «IF targetCard()»
 encode(o);
 return encoding_length;
 «ENDIF»
 «IF targetServer()||targetTerminal()»
 encode(o);
 return computeSize(o);
 «ENDIF»
}
 public short computeSize(«codeableName
   ()» o) {
 encoding_length = 0;
 try { compSize(o); return encoding_length; } catch
   (Exception e) { return -1; }
}
«IF targetCard()»
private void encodeInt(short o) {
 encodeDestination[encoding_length+
   +] = «codeName()».INT;
 Util
   .setShort(encodeDestination, encoding_length,
     (short)0x00);
 Util
   .setShort(encodeDestination, (short)
     (encoding_length+2), o);
 encoding_length += 4;
}«ENDIF»
public void encodeByte(byte o) {
 encodeDestination[encoding_length+
   +] = «codeName()».BYTE;
 encodeDestination[encoding_length++] = o;
}
private void encodeBoolean(boolean o) {
 encodeDestination[encoding_length++] = Code.BOOLEAN;
 encodeDestination[encoding_length+
   +] = (byte)( o ? 1 : 0 );
}
«IF targetCard()»
private void compSizeInt(short o) {
 encoding_length += 5;
 }
«ENDIF»
private void compSizeByte(byte o) {
 encoding_length += 2;
}
«IF targetServer()||targetTerminal()»
public void encodeInt(int i) {
 encodeDestination[encoding_length+
   +] = «codeName()».INT;
 ByteArray
   .setInt(encodeDestination, encoding_length, i);
 encoding_length += 4;
}
private void compSizeInt(int i) {
```

```
 encoding_length += 5;
}
public int decodeInt(byte[] in) {
 int res = ByteArray.getInt(in, ++encoding_length);
 encoding_length += 4;
 return res;
}
public byte[] decodeByteArray(byte[] in)
  throws «exceptionName()» {
 short len = ByteArray.getShort(in, +
   +encoding_length);
 byte[] dest = new byte[len];
 encoding_length += 2;
 if(!(len <= (in.length - encoding_length))) {
  stop();
 }
 ByteArray.copy(in, encoding_length, dest, 0, len);
 encoding_length += len;
 return dest;
}
public String byteToString(byte[] b) {
 int len = b.length;
 while(len > 0 && b[len-1] == 0) len--;
 return new String(ByteArray
   .subarray(b, 0, len), charset);
}
public String decodeString(byte[] in)
  throws «exceptionName()» {
 short len = ByteArray.getShort(in, +
   +encoding_length);
 byte[] dest = new byte[len];
 encoding_length += 2;
 if(!(len <= (in.length - encoding_length))) {
  stop();
 }
 ByteArray.copy(in, encoding_length, dest, 0, len);
 encoding_length += len;
 String str = byteToString(dest);
 return str;
}
«ENDIF»
private void compSizeBoolean(boolean o) {
 encoding_length += 2;
}
«IF targetCard()»
private short decodeInt(byte[] in) {
 if (in[encoding_length+
   +] != Code.INT) «abortClassName()»();
 short upper = Util.getShort(in, encoding_length);
 if(upper != 0x00){
  «abortClassName()»();
  return 0;
 }
 else
```

```
 {
  short lower = Util
    .getShort(in, (short)(encoding_length+2));
  encoding_length += 4;
  return lower;
 }
}
≪ENDIF≫
≪IF targetTerminal()≫
public byte[] encodeSingleString(String s) {
 byte[] o = stringToByte(s);
 byte[] dest = new byte[3 + o.length];
 int i = 0;
 dest[i++] = ≪codeName()≫.BYTEARRAY;
 ByteArray.setShort(dest, i, (short) (o.length));
 i += 2;
 ByteArray.copy(o, 0, dest, i, o.length);
 i += o.length;
 return dest;
}
≪ENDIF–≫
public void encodeByteArray(byte[] o) {
 ≪IF targetTerminal() || targetServer()≫
 ≪ENDIF≫
 encodeDestination[encoding_length+
   +] = ≪codeName()≫.BYTEARRAY;
 ≪IF targetTerminal() || targetServer()≫
 ByteArray
   .setShort(encodeDestination, encoding_length,
     (short)(o.length));
 ≪ENDIF–≫
 ≪IF targetCard()≫
 Util
   .setShort(encodeDestination, encoding_length,
     (short)(o.length));
 ≪ENDIF–≫
 encoding_length += 2;
 ≪IF targetTerminal() || targetServer()≫
 ByteArray
   .copy(o, 0, encodeDestination, encoding_length, o
     .length);
 ≪ENDIF–≫
 ≪IF targetCard()≫
 Util
   .arrayCopy(o,
     (short)0, encodeDestination, encoding_length,
       (short)o.length);
 ≪ENDIF–≫
 encoding_length += o.length;
}
private void compSizeByteArray(byte[] o) {
 encoding_length += 3;
 encoding_length += o.length;
}
```

```
 «IF targetCard()»
 private void decodeByteArrayInto
   (byte[] in, byte[] into) {
  if (in[encoding_length+
    +] != Code.BYTEARRAY) «abortClassName()»();
  short len = Util.getShort(in, encoding_length);
  encoding_length += 2;
  if (! (0 <= len && len <= into.length && len <= (in
    .length - encoding_length))) «abortClassName
      ()»();
  Util
    .arrayCopy(in, encoding_length, into,
      (short)0, len);
  if(len < into.length) Util
    .arrayFillNonAtomic(into, len, ((short)(into
      .length - len)), (byte)0);
  encoding_length += len;
 }
 «ENDIF»
«ENDDEFINE»

«DEFINE generateCodingForClass FOR Class»
 «IF this
   .isPrivateKey
     ()»«EXPAND coding4Privatekey
       FOR this-»
 «ELSEIF this
   .isPublicKey()»«EXPAND coding4Publickey
     FOR this-»
 «ELSEIF this
   .isEncDataSymm
     ()»«EXPAND coding4EncDataSymm
       FOR this.getModel()-»
 «ELSEIF this
   .isEncDataAsymm
     ()»«EXPAND coding4EncDataAsymm
       FOR this.getModel()-»
 «ELSEIF this
   .isListClass()»«EXPAND coding4List
     FOR this-»
 «ELSE-»
 «IF(targetServer()||targetTerminal())»
 public void encode«this.name.toString()
   .toFirstUpper()»(«this.name» c)
     throws «exceptionName()»{
  encodeDestination[encoding_length+
    +] = «codeName()».«this.name
      .toString().toUpperCase()»;
  «FOREACH this.getAttributesInCorrectOrder()
    AS attr-»
   «logging("General VON " + this.name + ": "
     + this.getGeneralClasses().size)»
   «logging("Type VON " + this.name + "."
     + attr.name + ": " + attr.translateType())»
```

```
  «IF attr.isPrimitiveType()||attr
    .isPrimitiveTypeArray()-»
   «IF attr
     .translateTypeForCodingMethod
       () == "String" && this
         .isSecurityDatatype()-»
    encodeByteArray(c.«attr.getterName()»());
   «ELSE-»
    encode«attr
      .translateTypeForCodingMethod()»((«attr
        .translateType()»)c.«attr
          .getterName()»());
   «ENDIF-»
  «ELSE-»
   encode(c.«attr.getterName()»());
  «ENDIF-»
 «ENDFOREACH-»
}
public «this.name.toString()» decode«this
  .name.toString()
    .toFirstUpper()»(byte[] in)
      throws «exceptionName()» {
 «this.name.toString()» c=new «this.name
   .toString()»();
 encoding_length++;
 «FOREACH this.getAttributesInCorrectOrder()
   AS attr-»
  «IF attr.isPrimitiveTypeArray()-»
   c.«attr.setterName()»(decode«attr
     .translateTypeForCodingMethod()»(in));
  «ELSEIF attr.isPrimitiveType()-»
   «IF attr
     .translateTypeForCodingMethod
       () == "String" && this
         .isSecurityDatatype() -»
    c.«attr
      .setterName()»(decodeByteArray(in));
   «ELSE-»
    c.«attr.setterName()»(decode«attr
      .translateTypeForCodingMethod()»(in));
   «ENDIF-»
  «ELSE-»
   c.«attr.setterName()»(((«attr
     .translateType()»)decode(in)));
  «ENDIF-»
 «ENDFOREACH-»
 return c;
}
private void compSize«this.name.toString()
  .toFirstUpper()»(«this.name» e)
    throws «exceptionName()»{
 encoding_length++;
 «FOREACH this.getAttributesInCorrectOrder()
   AS attr-»
```

```
  «IF attr.isPrimitiveType() || attr
    .isPrimitiveTypeArray()»
   «IF attr
     .translateTypeForCodingMethod
       () == "String" && this
         .isSecurityDatatype()-»
    compSizeByteArray(e.«attr
      .getterName()»());
   «ELSE-»
    compSize«attr
      .translateTypeForCodingMethod()»(e
        .«attr.getterName()»());
   «ENDIF-»
  «ELSE-»
   compSize(e.«attr.getterName()»());
  «ENDIF-»
 «ENDFOREACH-»
}
«ENDIF-»
«IF (targetCard())»
private void encode«this.name.toString()
  .toFirstUpper()»(«this.name» c) {
 encodeDestination[encoding_length+
   +] = «codeName()».«this.name
     .toString().toUpperCase()»;
 «FOREACH this.getAttributesInCorrectOrder()
   AS attr-»
  «IF attr.isPrimitiveType()||attr
    .isPrimitiveTypeArray()-»
   encode«attr
     .translateTypeForCodingMethod()»((«attr
       .translateType()»)c.«attr.name
         .toString()»);
  «ELSE-»
   encode(c.«attr.name.toString()»);
  «ENDIF-»
 «ENDFOREACH-»
}
private void decode«this.name.toString()
  .toFirstUpper()»Into(byte[] in, «this.name
    .toString()» into) {
 if(in[encoding_length+
   +] != Code.«this.name.toString()
     .toUpperCase()») «abortClassName()»
       ();
 «FOREACH this.getAttributesInCorrectOrder()
   AS attr-»
  «IF attr.isPrimitiveTypeArray()-»
   decode«attr
     .translateTypeForCodingMethod()»Into(in,into
       .«attr.name»);
  «ELSEIF attr.isPrimitiveType()-»
   into.«attr.name» = decode«attr
     .translateTypeForCodingMethod()»(in);
```

```
   «ELSE-»
    decode«attr
      .translateType()»Into(in, («attr
        .translateType()»)(into.«attr
          .name»));
   «ENDIF-»
  «ENDFOREACH-»
 }
 private void compSize«this.name.toString()
   .toFirstUpper()»(«this.name» d) {
  encoding_length++;
  «FOREACH this.getAttributesInCorrectOrder()
    AS attr-»
   compSize«IF attr.isPrimitiveType() || attr
     .isPrimitiveTypeArray()»«attr
       .translateTypeForCodingMethod
         ()»«ENDIF»(«IF attr
           .translateType()=="short"»
             (int)«ENDIF»d.«attr.name
               .toString()»); «ENDFOREACH-»
 }
 «ENDIF»
«ENDIF»
«ENDDEFINE»

«DEFINE coding4List FOR Class»
 private void encode«this.name»(«this
   .name» c) «IF targetServer
     () ||targetTerminal()»
       throws «exceptionName
         ()» «ENDIF» {
  encodeDestination[encoding_length+
    +] = Code.«this.name.toUpperCase()»;
  encodeDestination[encoding_length+
    +] = (byte)c.index();
  for(int i=0; i<c.index(); i++)
   encode(c.elems[i]);
 }
 «IF targetCard()»
 private void decode«this
   .name»Into(byte[] in, «this
     .name» into) {
  if (in[encoding_length+
    +] != Code.«this.name.toUpperCase()»)
    «abortClassName()»();
  byte index=in[encoding_length++];
  if(index>into.size())
   «abortClassName()»();
  into.index=index;
  for(int i=0; i<index; i++)
   decode«this.getListElemTyp()
     .name»Into(in, into.elems[i]);
 }
«ENDIF»
```

```
«IF targetServer() ||targetTerminal()»
private «this.name» decode«this
  .name»(byte[] in)
    throws «exceptionName()»{
 if (in[encoding_length+
   +] != Code.«this.name.toUpperCase()»)
   «abortName()»();
 byte index=in[encoding_length++];
 short size=«this.getAllAttributes()
   .selectFirst(a|a.name == "elems").upper
     .toString()»;
 if(index>size)
  «abortName()»();
 «this.name» lt=new «this.name»(size);
 for(int i=0; i<index; i++)
  lt.add((«this.getListElemTyp()
    .name»)decode(in));
 return lt;
}
«ENDIF»
private void compSize«this.name»(«this
  .name» c) «IF targetServer
    () ||targetTerminal()»
      throws «exceptionName
        ()» «ENDIF»{
 encoding_length++;
 encoding_length++;
 for(int i=0; i<c.index(); i++)
  compSize(c.elems[i]);
}
«ENDDEFINE»

«DEFINE coding4Publickey FOR Class»
private void encodePublicKey
  (PublicKey c) «IF targetServer
    () ||targetTerminal()»
      throws «exceptionName
        ()» «ENDIF»{
 encodeDestination[encoding_length+
   +] = Code.PUBLICKEY;
 «IF targetServer()||targetTerminal
   () » encodeByteArray(c
     .getModulus());«ENDIF»
 «IF targetCard()» encodeByteArray(c
   .modulus);«ENDIF»
}
«IF targetCard()»
private void decodePublicKeyInto
  (byte[] in, PublicKey into) {
 if (in[encoding_length+
   +] != Code.PUBLICKEY) «abortClassName()»();
 decodeByteArrayInto(in, into.modulus);
 into.updateKey();
}«ENDIF»
```

```
 ≪IF targetServer() ||targetTerminal()≫
 public PublicKey decodePublicKey(byte[] in)
   throws ≪exceptionName()≫{
  if (in[encoding_length+
    +] != Code.PUBLICKEY) ≪abortName()≫();
  PublicKey res = new PublicKey();
  res.setKey(decodeByteArray(in));
  return res;
 }≪ENDIF≫
  private void compSizePublicKey
    (PublicKey c) ≪IF targetServer
      () ||targetTerminal()≫
        throws ≪exceptionName
          ()≫ ≪ENDIF≫{
  encoding_length++;
  ≪IF targetServer()||targetTerminal
    ()≫ compSizeByteArray(c
      .getModulus());≪ENDIF≫
  ≪IF targetCard()≫compSizeByteArray(c
    .modulus);≪ENDIF≫
 }
≪ENDDEFINE≫

≪DEFINE coding4Privatekey FOR Class≫
 public void encodePrivateKey
   (PrivateKey c) ≪IF targetServer
     () ||targetTerminal()≫
       throws ≪exceptionName
         ()≫ ≪ENDIF≫{
  encodeDestination[encoding_length+
    +] = Code.PRIVATEKEY;
  ≪IF targetServer()||targetTerminal()≫
  encodeByteArray(c.getModulus());
  encodeByteArray(c.getPrivexponent());≪ENDIF≫
  ≪IF targetCard()≫
  encodeByteArray(c.modulus);
  encodeByteArray(c.privexponent);≪ENDIF≫
 }
 ≪IF targetServer()||targetTerminal()≫
 public PrivateKey decodePrivateKey(byte[] in)
   throws ≪exceptionName()≫{
  if (in[encoding_length+
    +] != Code.PRIVATEKEY) ≪abortName()≫();
  PrivateKey res = new PrivateKey();
  res.setKey(decodeByteArray(in),decodeByteArray(in));
  return res;
 }≪ENDIF≫
 ≪IF targetCard()≫
 private void decodePrivateKeyInto
   (byte[] in, PrivateKey into) {
  if (in[encoding_length+
    +] != Code
      .PRIVATEKEY) ≪abortClassName()≫();
  decodeByteArrayInto(in, into.modulus);
```

```
  decodeByteArrayInto(in, into.privexponent);
  into.updateKey();
 }≪ENDIF≫
  private void compSizePrivateKey
    (PrivateKey c) ≪IF targetServer
      () ||targetTerminal()≫
        throws ≪exceptionName
          ()≫ ≪ENDIF≫{
  encoding_length++;
  ≪IF targetServer()||targetTerminal()≫
  compSizeByteArray(c.getModulus());
  compSizeByteArray(c.getPrivexponent());
  ≪ENDIF≫
  ≪IF targetCard()≫
  compSizeByteArray(c.modulus);
  compSizeByteArray(c.privexponent);
  ≪ENDIF≫
 }
≪ENDDEFINE≫

≪DEFINE coding4EncDataSymm FOR Model≫
 private void encodeEncDataSymm(EncDataSymm c) {
  encodeDestination[encoding_length++] = Code.ENCDAT
    ASYMM;
  encodeByteArray(c.encrypted);
  ≪IF targetCard()≫
  encodeInt(c.plainlength);
  encodeInt(c.enclength);
  ≪ENDIF≫
  ≪IF targetServer()||targetTerminal()≫
  encodeInt(c.plainlength);
  encodeInt(c.enclength);
  ≪ENDIF≫
  encodeByte(c.plainDataType);
 }
 ≪IF targetCard()≫
 private void decodeEncDataSymmInto
   (byte[] in, EncDataSymm into)≪IF targetServer
     () ||targetTerminal()≫
       throws ≪exceptionName
         ()≫ ≪ENDIF≫ {
  if (in[encoding_length++] != Code.ENCDAT
    ASYMM) ≪abortClassName()≫();
  decodeByteArrayInto(in, into.encrypted);
  into.plainlength = decodeInt(in);
  into.enclength = decodeInt(in);
  byte b = decodeByte(in);
  into.plainDataType = b;
  if(≪FOREACH this.getAllPlainDataClasses()
    AS c SEPARATOR " && "≫b != ≪codeName
      ()≫.≪c.name.toString()
        .toUpperCase
          ()≫≪ENDFOREACH≫) ≪abortClassName
            ()≫();
```

```
 }
 «ENDIF»
 «IF targetServer()||targetTerminal()»
 public EncDataSymm decodeEncDataSymm(byte[] in)
   throws «exceptionName()»{
  if (in[encoding_length++] != Code.ENCDAT
    ASYMM) «abortName()»();
  EncDataSymm into = new EncDataSymm();
  into.encrypted = decodeByteArray(in);
  into.plainlength = decodeInt(in);
  into.enclength = decodeInt(in);
  into.plainDataType = decodeByte(in);
  return into;
 }
 «ENDIF»
 private void compSizeEncDataSymm(EncDataSymm c) {
  encoding_length++;
  compSizeByteArray(c.encrypted);
  encoding_length += 12;
 }
«ENDDEFINE»

«DEFINE coding4EncDataAsymm FOR Model»
 private void encodeEncDataAsymm(EncDataAsymm c) {
  encodeDestination[encoding_length++] = Code.ENCDATA
    ASYMM;
  encodeByteArray(c.encrypted);
  «IF targetCard()»
  encodeInt(c.plainlength);
  encodeInt(c.enclength);
  «ENDIF»
  «IF targetServer()||targetTerminal()»
  encodeInt(c.plainlength);
  encodeInt(c.enclength);
  «ENDIF»
  encodeByte(c.plainDataType);
 }
«IF targetCard()»
 private void decodeEncDataAsymmInto
   (byte[] in, EncDataAsymm into) {
  if (in[encoding_length++] != Code.ENCDATA
    ASYMM) «abortClassName()»();
  decodeByteArrayInto(in, into.encrypted);
  into.plainlength = decodeInt(in);
  into.enclength = decodeInt(in);
  byte b = decodeByte(in);
  into.plainDataType = b;
  if(«FOREACH this.getAllPlainDataClasses()
    AS c SEPARATOR " && "»b != «codeName
      ()».«c.name.toString()
        .toUpperCase
          ()»«ENDFOREACH») «abortClassName
            ()»();
 }
```

```
 «ENDIF»
 «IF targetServer()||targetTerminal()»
 public EncDataAsymm decodeEncDataAsymm
   (byte[] in) throws «exceptionName()» {
  if (in[encoding_length++] != Code.ENCDATA
    ASYMM) «abortName()»();
  EncDataAsymm into = new EncDataAsymm();
  into.encrypted = decodeByteArray(in);
  into.plainlength = decodeInt(in);
  into.enclength = decodeInt(in);
  into.plainDataType = decodeByte(in);
  return into;
 }
 «ENDIF»
 private void compSizeEncDataAsymm(EncDataAsymm c) {
  encoding_length++;
  compSizeByteArray(c.encrypted);
  encoding_length += 12;
 }
«ENDDEFINE»

«DEFINE generateCodingFooter FOR Model»
   «IF targetServer()||targetTerminal()»
   private void «abortName()»()
     throws «exceptionName()»{
    throw new «exceptionName()»
      ("«codingName()»");
   }«ENDIF»
}
«ENDDEFINE»

 «DEFINE generateCodingMain(Model mo)
   FOR List[Class]»
 public void encode(«codeableName
   ()» c) «IF targetServer()||targetTerminal
     ()»
       throws «exceptionName()»«ENDIF»{
  switch(c.getCode()) {
  «FOREACH this AS class-»
   case «codeName()».«class.name
     .toString().toUpperCase()»:
    encode«class.name.toString()
      .toFirstUpper()»((«class.name»)c);
   break;
  «ENDFOREACH-»
   default :
   «IF targetServer()||targetTerminal()»
    «abortName()»();«ENDIF»
   «IF targetCard()»
   «abortClassName()»();
   «ENDIF»
  }
 }
 «IF targetCard()»
```

```
public «mo
  .getNameMessageSuperclass()» decodeMessage
    (byte[] in, short offset, short expectedLength) {
 encoding_length = offset;
 «mo.getNameMessageSuperclass()» m = null;
 try {
 switch(in[encoding_length]) {
 «FOREACH this.select(e|e
   .isMessage() && isUsedCodeableClass(e) && !e
     .isAbstract()) AS class-»
  case «codeName()».«class.name
    .toString().toUpperCase()»:
   m = Store.new«class.name.toString()»();
   decode«class.name.toString()
     .toFirstUpper()»Into(in, («class.name
       .toString()»)m);
   break;
 «ENDFOREACH-»
 }
 }
 catch
   (ArrayIndexOutOfBoundsException e) { «abortClassName
     ()»(); }
 if (m == null || encoding_length != (short)
   (expectedLength
     + offset)) «abortClassName()»();
 return m;
}
«ENDIF»
«IF targetServer()||targetTerminal()»
public «codeableName()» decode
  (byte[] in) throws «exceptionName()»{
 try {
 switch(in[encoding_length]) {
 «FOREACH this AS class-»
  case «codeName()».«class.name
    .toString().toUpperCase()»:
   return decode«class.name.toString()
     .toFirstUpper()»(in);
 «ENDFOREACH-»
  default :
   «abortName()»();
 }
 }
 catch(ArrayIndexOutOfBoundsException e){
  «abortName()»();
 }
 return null;
}
«ENDIF»
public void compSize(«codeableName
  ()» c) «IF targetServer()||targetTerminal
    ()»
      throws «exceptionName
```

```
        ()≫ ≪ENDIF≫ {
 switch(c.getCode()) {
 ≪FOREACH this AS class-≫
  case ≪codeName()≫.≪class.name
    .toString().toUpperCase()≫:
   ≪IF targetServer()||targetTerminal()≫
   compSize≪class.name.toString()
     .toFirstUpper()≫((≪class
       .name≫)c);≪ENDIF≫
   ≪IF targetCard()≫
   compSize≪class.name.toString()
     .toFirstUpper()≫((≪class.name
       .toString()≫)c);≪ENDIF≫
  break;
 ≪ENDFOREACH-≫
  default :
  ≪IF targetServer()||targetTerminal()≫
   ≪abortName()≫();
  ≪ENDIF≫
  ≪IF targetCard()≫
  ≪abortClassName()≫();
  break;
  ≪ENDIF≫
 }
}
≪IF mo.requiresEncryption()≫
public PlainData decodePlainData
  (byte[] in, byte expectedType) ≪IF targetServer
    ()||targetTerminal()≫
      throws ≪exceptionName
        ()≫ ≪ENDIF≫{
 encoding_length = 0;
 PlainData p = null;
 try {
 if
   (in[encoding_length] != expectedType) {≪IF targetServer
     () ||targetTerminal()≫ ≪abortName()≫
       (); ≪ENDIF≫
 ≪IF targetCard() ≫
 ≪abortClassName()≫();≪ENDIF≫}
 switch(in[encoding_length]) {
 ≪IF targetCard() ≫
 ≪FOREACH mo.getCodeableClasses().select(e|e
   .isPlainData()) AS class-≫
  case ≪codeName()≫.≪class.name
    .toString().toUpperCase()≫:
   p = Store.new≪class.name.toString()≫();
   decode≪class.name.toString()
     .toFirstUpper()≫Into(in, (≪class.name
       .toString()≫)p);
   break;
 ≪ENDFOREACH-≫≪ENDIF≫
 ≪IF targetTerminal()||targetServer()≫
 ≪FOREACH mo.getCodeableClasses().select(e|e
```

```
    .isPlainData()) AS class-≫
   case ≪codeName()≫.≪class.name
     .toString().toUpperCase()≫:
    p = decode≪class.name.toString()
      .toFirstUpper()≫(in);
    break;
  ≪ENDFOREACH-≫≪ENDIF≫
  }
  }
  catch(ArrayIndexOutOfBoundsException e) {
  ≪IF targetServer() ||targetTerminal
    ()≫ ≪abortName()≫(); ≪ENDIF≫
  ≪IF targetCard() ≫
  ≪abortClassName()≫();≪ENDIF≫
  }
  if (p == null) {
  ≪IF targetServer() ||targetTerminal
    ()≫ ≪abortName()≫();≪ENDIF≫
  ≪IF targetCard() ≫
  ≪abortClassName()≫();≪ENDIF≫
  }
  return p;
 }
 ≪ENDIF≫
≪ENDDEFINE≫
```

3.3.3 Getter-Operations

```
import uml;
extension psm_templates_shared::names;
extension psm_templates_shared::parse;
extension psm_templates_shared::tests;
extension psm_templates_shared::translations;
extension org::eclipse::xtend::util::stdlib::elementprops;

String pp(List[String] strs) :
 (strs.size == 0 ? "" :
 (strs.size == 1 ? strs.first() :
 strs.first() + ", " + pp(strs.withoutFirst())
 ));

String ppListBetween
  (List[String] strs, String between) :
 (strs.size == 0 ? "" :
 (strs.size == 1 ? strs.first() :
 strs.first() + between + ppListBetween
   (strs.withoutFirst(), between)
 ));

String ppImplements(Model mo) :
 (let imps = mo.getUsedPredefinedInterfaces() :
 (imps.size
   == 0
     ? "" : " implements " + imps.ppListBetween(", "))
 );
```

```
List[NamedElement] removeDoubles
  (List[NamedElement] l) :
 ( l.isEmpty ? l :
 ( let first = l.first() :
 ( let rest = l.withoutFirst() :
 ( rest.exists(n | n
   == first)
     ? rest.removeDoubles() : rest.removeDoubles().add
       (first) )
 )));

List[String] removeStringDoubles(List[String] l) :
 ( l.isEmpty ? l :
 ( let first = l.first() :
 ( let rest = l.withoutFirst() :
 ( rest.exists(n | n.matches(first))
   ? rest.removeDoubles() : rest.removeStringDoubles
     ().add(first) )
 )));
cached List[Class] getUsedClasses(Class cla) :
 let usages = cla.getModel().allOwnedElements
   ().typeSelect(Usage) :
 let matches = usages.select(e|e.client.first
   ().name.toString().matches(cla.name.toString())) :
 let res = matches.supplier.typeSelect(Class) :
 res
 ;
cached Class getCertificateDataClass(Class cla) :
 let atttype = cla.getAttributesInCorrectOrder
   ().type.select(e|e.name!="SignedData"):
  ((atttype.isEmpty)? null : atttype.first());
cached List[String] getCustomMethods(Class cla) :
   if (targetCard()) then{
 let m1 = ((cla.getModel().getCards().reject
   (e|e.isAbstract()).contains(cla))
     ? {"process"} : ((List[String]){})) :
 m1}
 else
 null;
cached UnlimitedNatural getListUpper(Class c) :
 c.getAllAttributes().selectFirst(e|e.getUpper
   () != 1).getUpper();
cached List[Class] getAllTerminalClasses(Model mo) :
 ((List[Class])mo.getMessages().addAll(mo.getTerminals
   ())).getClassesRec({ });
cached List[Class] getAllTerminalClasses
  (Model mo, Class terminal) :
 ((List[Class])mo.getMessages().addAll
   (terminal.getGeneralClasses()).add
     (terminal)).getClassesRec({ });
cached List[Class] getAllSmartcardClasses(Model mo) :
 let res = ((List[Class])mo.getMessages().addAll
   (mo.getCards())).getClassesRec({ }) :
 res
```

```
 ;

List[Class] getAllSmartcardClasses
  (Model mo, Class card) :
 let res1 = ((List[Class])mo.getMessages().addAll
   (card.getGeneralClasses()).add(card)) :
 let res = res1.getClassesRec({ }) :
 res
 ;
cached List[Class] getAllServerClasses(Model mo) :
 ((List[Class])mo.getMessages().addAll(mo.getServers
   ())).getClassesRec({ });
cached List[Class] getAllServerClasses
  (Model mo, Class server) :
 ((List[Class])mo.getMessages().addAll
   (server.getGeneralClasses()).add
     (server)).getClassesRec({ });
cached List[Class] getAllClasses(Model mo) :
 if(targetServer()) then{
  getAllServerClasses(mo)
  }
 else if(targetTerminal()) then
  getAllTerminalClasses(mo)
 else if(targetCard()) then
  getAllSmartcardClasses(mo);
cached Package getDeploymentDiagram(Model m) :
m.packagedElement.typeSelect(Package).selectFirst
   (e|e.hasStereotype("SecureMDD::DeploymentDiagram"));

List[Class] getClassesRec
  (List[Class] todo, List[Class] done) :
 todo.isEmpty ? done :
  (let c = todo.first() :
   done.contains(c)
     ? todo.withoutFirst().getClassesRec(done) :
   (let newcs1 = c.ownedAttribute.collect
     (p|p.type).toSet().typeSelect(Class) :
   let newcs2 = c.getUsedClasses().reject
     (e|e.hasStereotype("SecureMDD::Message")) :
   let newcs3 = c.general.typeSelect(Class) :
    ((List[Class])todo.withoutFirst().addAll
      (newcs1).addAll(newcs2).addAll
        (newcs3)).getClassesRec((List[Class])done.add
          (c))
   )
  )
    ;

Set[Class] getCDClassesForInitialization(Class card) :
 (targetCard() ?
  card.getConstructor() :
  card.getModel().getTerminals().reject(e|e.isAbstract
    ()).first().ownedOperation.select(e|e.name.matches
      ('initCard_'+card.name)).first()
```

```
 ).ownedParameter.type.reject(e|!card.getModel
   ().getClassDiagramClasses().contains(e));

Set[Class] getAllClassesForInitialization(Class card) :
 (targetCard() ?
  card.getConstructor() :
  card.getModel().getTerminals().reject(e|e.isAbstract
    ()).first().ownedOperation.select(e|e.name.matches
      ('initCard_'+card.name)).first()
 ).ownedParameter.type.typeSelect(Class);

List[Class] getCDClassesForInitialization(Model mo) :
 (targetCard() ?
  mo.getCards().reject(e|e.isAbstract
    ()).getCDClassesForInitialization().reject(e|e
      ==null) :
  mo.getTerminals().ownedOperation.select
    (e|e.name.startsWith
      ('initCard_')).ownedParameter.type.reject
        (e|!mo.getClassDiagramClasses().contains(e))
 );

List[Class] getAllClassesForInitialization(Model mo) :
 (targetCard()
  ? mo.getCards().reject(e|e.isAbstract
    ()).getAllClassesForInitialization().reject(e|e
      ==null) :
 (targetTerminal()
   ? mo.getTerminals().ownedOperation.select
     (e|e.name.startsWith
       ('initCard_')).ownedParameter.type.typeSelect
         (Class)
   : {}
   )
 ).removeDoubles();
cached List[Class] getCodeableClasses(Model mo) :
 let clas = ((List[Class])(mo.getMessages
   ())).getClassesRec({ }) :
 let clas1 = clas.addAll
   (mo.getAllClassesForInitialization().reject
     (e|clas.contains(e))) :
 let clas7 = ( targetCard()
     ? clas1.addAll(mo.getCards().getClassesRec
       ({}).select(e|e.isCryptoClass
         () && !clas1.contains(e)))
     : (targetTerminal()
      ? clas1.addAll(mo.getTerminals().getClassesRec
        ({}).select(e|e.isCryptoClass
          () && !clas1.contains(e)))
      : clas1.addAll(mo.getServers().getClassesRec
        ({}).select(e|e.isCryptoClass
          () && !clas1.contains(e))))
   ) :
 clas7;
```

```
cached List[Class] getCodeableClasses
  (Model mo, Class terminalcard) :
 let clas00 = mo.getMessages() :
 let clas = (terminalcard.isTerminal()
  ? ((List[Class])clas00.addAll
    (mo.getAllTerminalClasses(terminalcard)) )
  : ((List[Class])clas00.addAll
    (mo.getAllSmartcardClasses(terminalcard)) )
 ) :
 let clas10 = clas.select(e|e.isMessage()) :
 let clas0 = clas10.getClassesRec({ }) :
 let clas1 = clas0.addAll
   (mo.getAllClassesForInitialization().reject
     (e|clas0.contains(e))) :
 let clas7 = clas1.addAll(((List[Class]){}.add
   (terminalcard)).getClassesRec({}).select
     (e|e.isCryptoClass() && !clas1.contains(e))) :
 clas7;

List[Class] getAllPlainDataClasses(Model mo) :
 mo.getCodeableClasses().select(c|c.isPlainData());

List[Class] getNormalGeneratedClasses
  (Model mo, Class terminalcard) :
 if(terminalcard.isTerminal()) then
  mo.getAllTerminalClasses(terminalcard).reject
    (c|mo.getTerminals().contains
      (c) || c.isSecurityDatatype() || c.hasStereotype
        ("SecureMDD::Manual")).reject
          (e|getCustomTerminalClasses().contains
            (e.name.toString()) || e.isListClass() )
 else
  if(terminalcard.isCard()) then
   mo.getAllSmartcardClasses(terminalcard).reject
     (c|mo.getCards().contains
       (c) || c.isSecurityDatatype
         () || c.hasStereotype
           ("SecureMDD::Manual")).reject
             (e|getCustomCardClasses().contains
               (e.name.toString()) || e.isListClass() )
  else if(terminalcard.isServer()) then
   mo.getAllServerClasses(terminalcard).reject
     (c|mo.getServers().contains
       (c) || c.isSecurityDatatype
         () || c.hasStereotype
           ("SecureMDD::Manual")).reject
             (e|getCustomServerClasses().contains
               (e.name.toString()) || e.isListClass() )
  else
   null
;

List[Class] getClassDiagramClassesAndSecurityDatatypeClasses
  (Model mo) :
```

```
 mo.getSecurityDatatypesClasses().union
   (mo.getClassDiagramClasses());

List[String] getCustomCardClassAttributes(Class cla) :
(
 ( cla.getModel().getCards().contains(cla) ) ?
   ( (List[String]){}
    .add("coding") ) :
 (
 ( cla.getModel().getCards().contains(cla) ) ?
  ( (List[String]){}
    .add("coding")) :
 (
 ( cla.getModel().getCards().contains(cla) ) ?
  ( (List[String]){}
    .add("coding")) :
 (
 (List[String]){})
 ))
);

List[String] getCustomCardClasses() :
 (List[String]){}
  .add(mathName())
  .add(storeName())
  .add(simpleCommName())
  .add(codingName())
  .add(codeName())
  .add("javacard.framework.Applet")
 ;

List[String] getCustomTerminalClasses() :
 (List[String]){}
  .add(deploymentPropertiesName())
  .add(codingName())
  .add(lengthConstantsName())
  .add(terminalExceptionName())
  .add(codeName())
  .add(messageWrapperName())
 ;

List[String] getCustomServerClasses() :
 (List[String]){}
  .add(deploymentPropertiesName())
  .add(codingName())
  .add(lengthConstantsName())
  .add(terminalExceptionName())
  .add(codeName())
  .add(messageWrapperName())
 ;

List[Class] getUserMessageClasses
  (Model mo, Class terminal) :
 mo.packagedElement.typeSelect
```

```
    (Package).ownedElement.typeSelect(Class)
  .select(e|e.general.exists(f|f.hasStereotype
    ("SecureMDD::Usermessage")))
  .addAll(mo.getUserMessages());

List[Class] getUserMessageClasses(Model mo) :
 mo.packagedElement.typeSelect
    (Package).ownedElement.typeSelect(Class)
  .select(e|e.general.exists(f|f.hasStereotype
    ("SecureMDD::Usermessage")))
  .addAll(mo.getUserMessages());

List[Class] getUserMessages(Model mo) :
 mo.ownedElement.typeSelect
    (Package).ownedElement.typeSelect(Class).select
      (e|e.hasStereotype("SecureMDD::Usermessage"));

Class getUserMessage(Model mo) :
 mo.ownedElement.typeSelect
    (Package).ownedElement.typeSelect(Class).select
      (e|e.hasStereotype
        ("SecureMDD::Usermessage")).first();

Class getUserMessage(Message m, Model mo) :
 mo.getUserMessageClasses().selectFirst(e|e.name
    == m.name.getSeqSendClass());
cached List[Message] getSentMessagesFrom
   (Interaction seq, Class termcard) :
 seq.ownedElement.typeSelect(Message).select(msg|
  msg.sentBy(termcard)
 ).reject(msg|msg.isReceivedBy(termcard));

List[Class] getSentUserMessages(Class termcard) :
 termcard.getModel().getAllSequences
    ().ownedElement.typeSelect(Message).select(m|(
      (MessageOccurrenceSpecification)m.receiveEvent).covered.exists
        (l|l.isUser()))
   .select(m|m.sentBy(termcard)).getUserMessage
     (termcard.getModel());

List[Class] getReceivedUserMessages(Class termcard) :
 termcard.getModel().getAllSequences
    ().ownedElement.typeSelect(Message).select(m|(
      (MessageOccurrenceSpecification)m.sendEvent).covered.exists
        (l|l.isUser()))
   .select(m|m.isReceivedBy(termcard)).getUserMessage
     (termcard.getModel());

List[Property] getConnectedTerminals
   (Interaction sequence, Property cardSeqAttr, Model moTerm) :
 ((List[MessageOccurrenceSpecification])
  sequence.getMessagesToTerminal
    (cardSeqAttr, moTerm).receiveEvent)
 .covered.represents.removeDoubles();
```

```
List[Message] getMessagesToTerminal
  (Interaction sequence, Property cardSeqAttr, Model moTerm) :
 (let ms = sequence.ownedElement.typeSelect
   (Lifeline).selectFirst(l|l.represents
     == cardSeqAttr).coveredBy.typeSelect
       (MessageOccurrenceSpecification) :
 (let res =  ms.select(mo|mo.message.sendEvent == mo
   && (
     (MessageOccurrenceSpecification)mo.message.receiveEvent).covered.exists
       (c| moTerm.getTerminals().exists(e| e.name
         == c.represents.type.name)))
  .message :
  res));

Property getReceiveAttr(Message msg) :
 ((MessageOccurrenceSpecification)msg.receiveEvent).
      covered.first().represents;
cached Package getClassDiagram(Model m) :
 m.packagedElement.typeSelect(Package).select
   (e|e.hasStereotype("SecureMDD::ClassDiagram"));
cached Package getSecurityDatatypes(Model m) :
 m.packagedElement.typeSelect(Package).select
   (e|e.hasStereotype("SecureMDD::SecurityDatatypes"));
cached List[Class] getClassDiagramClasses(Model m) :
 m.getClassDiagram().ownedElement.typeSelect(Class);
cached List[Class] getTestcaseClasses(Model m):
 m.getClassDiagram().ownedElement.typeSelect
   (Package).select(e|e.name
     =="Testcase").ownedElement.typeSelect(Class);
cached List[Class] getTestcaseInterfaces(Model m):
 m.getClassDiagram().ownedElement.typeSelect
   (Package).select(e|e.name
     =="Testcase").ownedElement.typeSelect(Interface);
cached Class getTestcaseClass(Model m, String cla):
 m.getTestcaseClasses().typeSelect(Class).selectFirst
   (e|e.name.toLowerCase() == cla.toLowerCase());
cached List[Class] getSecurityDatatypesClasses
  (Model m) :
 m.getSecurityDatatypes().ownedElement.typeSelect
   (Class);
cached Property getStateAttribute(Class this) :
 this.getAllAttributesG().selectFirst
   (a|a.isStateAttribute());

List[Operation] instanceOperations(Class this) :
 this.ownedOperation;
Boolean hasConstructor(Class this) :
 (let ops = this.instanceOperations() :
  ops.exists(op | op.name.toString()
    == this.name.toString()));

Operation getConstructor(Class this) :
 (let ops = this.instanceOperations() :
  ops.selectFirst(op | op.name.toString()
```

```
    == this.name.toString
      () && !op.ownedParameter.isEmpty)
  );

List[Property] instanceAttributesFromAssociation
  (Class this) :
 this.ownedAttribute.select(e|e.association != null);

List[Property] instanceAttributesWithoutAssociation
  (Class this) :
 this.ownedAttribute.select(e|e.association == null);

List[Property] instanceAttributes(Class this) :
 this.instanceAttributesWithoutAssociation().addAll
   (this.instanceAttributesFromAssociation());

List[Property] getAttributesInCorrectOrder
  (Class this) :
 ( this.hasConstructor()
   ? this.getAttributesFromConstructor
     () : this.getAllAttributesG() );

List[Property] getAllAttributesG(Class this) :
 ( this.getGeneralClasses().getAllAttributes().addAll
   (this.instanceAttributes()) );

List[Property] getAttributesFromConstructor
  (Class this) :
 (let op = this.getConstructor() :
 (let params = op.ownedParameter :
  params.collect(p| this.getAttributeWithName
    (p.name.toString()))
 ));

Property getAttributeWithName(Class this, String n) :
 let att = this.ownedAttribute.selectFirst
   (a|a.name.toString() == n) :
  if(att
    ==null) then this.getGeneralClasses
      ().ownedAttribute.selectFirst(a|a.name.toString
        () == n) else att ;

Property getPropertyForParam
  (Class this,Parameter param) :
 this.getAllAttributesG().select(e|e.name.toString()
   ==param.name.toString()).first();
cached List[Activity] getAllActivities(Model m) :
 m.ownedElement.typeSelect(Package)
 .select(e|e.hasStereotype
   ("SecureMDD::ActivityDiagram"))
 .ownedElement.typeSelect(Activity);
cached List[Activity] getAllTestcases(Model m) :
 m.ownedElement.typeSelect(Package)
 .select(e|e.hasStereotype
```

```
    ("SecureMDD::TestcaseDiagram"))
  .ownedElement.typeSelect(Activity).select
    (e|e.ownedElement.exists(e|e.metaType
      == ActivityPartition));

List[Class] getAllConstraintClasses(Model m) :
 m.getTestcaseClasses().select(c|
  m.getAllTestcases().ownedElement.typeSelect
     (Constraint).exists(constr|constr.getClassName()
       == c.name)
 );

Constraint getCorrespondingConstraint(Class cla) :
 cla.getModel().getAllTestcases
    ().ownedElement.typeSelect(Constraint).selectFirst
      (constr|cla.name == constr.getClassName());
cached String getClassName(Constraint c) :
 ((Activity)c.eContainer).name.toFirstUpper
    () + "_" + c.name.toFirstUpper() + "Constraint";
cached List[Interaction] getAllSequences(Model m) :
 m.ownedElement.typeSelect(Package)
 .select(e|e.hasStereotype
    ("SecureMDD::SequenceDiagram"))
 .ownedElement.typeSelect
    (Collaboration).ownedElement.typeSelect
      (Interaction);
cached List[Interaction] getAllTestcaseSequences
  (Model m) :
 m.ownedElement.typeSelect(Package)
 .select(e|e.hasStereotype
    ("SecureMDD::TestcaseDiagram"))
 .ownedElement.typeSelect
    (Collaboration).ownedElement.typeSelect
      (Interaction);

Interaction getSequence(Model m, String s) :
 if(m.getAllSequences().exists(e|e.name == s)) then
  m.getAllSequences().selectFirst(e|e.name == s)
 else
  m.getAllTestcaseSequences().selectFirst
     (seq|seq.name == s).redefinedBehavior.first();

Interaction getRealSequence(Interaction s) :
 s.redefinedBehavior.first();

Class receivedBy(Message m) :
 ((MessageOccurrenceSpecification)m.receiveEvent).
             covered.first().represents.type;

Property receivedByLifelineProp(Message m) :
 ((MessageOccurrenceSpecification)m.receiveEvent).covered.first().represents;

String getMessageName(Message m) :
 m.name.split("(").first().trim();
```

```
cached List[Enumeration] getAllEnumerations(Model m) :
 m.packagedElement.typeSelect
    (Package).ownedElement.typeSelect(Enumeration);

cached getMessages(Model m) :
 m.getClassDiagramClasses().select(c|c.isMessage());

List[Class] getTerminals(Model m) :
 (let res = m.getClassDiagramClasses().select
    (e| e.hasStereotype("SecureMDD::Terminal")
      || e.getGeneralTerminal() != null
      || e.hasStereotype("SecureMDD::PC")) :
  res);

List[Class] getCards(Model m) :
 m.getClassDiagramClasses().
 select(e|(e.hasStereotype("SecureMDD::Smartcard")
         || e.getGeneralCard() != null)
         );

List[Class] getServers(Model m) :
 m.getClassDiagramClasses().
 select(e|e.hasStereotype
    ("SecureMDD::Service") || e.getGeneralServer
      () != null || e.hasServerSubclasses() );
cached Class getNonce(Model m) :
 m.getSecurityDatatypesClasses().select(e|e.isNonce()).
 first();
cached Class getSecret(Model m) :
 m.getSecurityDatatypesClasses().select(e|e.isSecret())
 .first();
cached Class getKey(Model m) :
 m.getSecurityDatatypesClasses().
 select(e|e.isKey()).
 first();
cached Class getSymmkey(Model m) :
 m.getSecurityDatatypesClasses().
 select(e|e.isSymmkey()).
 first();
cached Class getAsymmkey(Model m) :
 m.getSecurityDatatypesClasses().
 select(e|e.isAsymmkey()).
 first();
cached Class getPrivateKey(Model m) :
 m.getSecurityDatatypesClasses().
 select(e|e.isPrivateKey()).
 first();
cached Class getPublicKey(Model m) :
 m.getSecurityDatatypesClasses().
 select(e|e.isPublicKey()).
 first();
cached Class getEncData(Model m) :
 m.getSecurityDatatypesClasses().select(e|e.isEncData
    ()).first();
```

```
cached Class getEncDataSymm(Model m) :
 m.getSecurityDatatypesClasses().select
    (e|e.isEncDataSymm()).first();
cached Class getEncDataAsymm(Model m) :
 m.getSecurityDatatypesClasses().select
    (e|e.isEncDataAsymm()).first();
cached Class getSignedData(Model m) :
 m.getSecurityDatatypesClasses().select
    (e|e.isSignedData()).first();
cached Class getHashedData(Model m) :
 m.getSecurityDatatypesClasses().select
    (e|e.isHashedData()).first();
cached Class getMACData(Model m) :
 m.getSecurityDatatypesClasses().select(e|e.isMACData
    ()).first();
cached List[Class] getAllConstants(Model m) :
 m.getClassDiagramClasses().select(e|e.isConstants());
cached List[Class] getStarAssociations(Model m) :
 let ass = m.getClassDiagram().ownedElement.typeSelect
    (Association) :
  ass.select(a| a.memberEnd.exists
     (e|e.upper > 1 || e.upper==-1))
      .collect(a|a.memberEnd.select
        (e|e.upper > 1 || e.upper==-1).first().type)
 ;
List [Class] getListClasses(Model m) :
 m.getClassDiagramClasses().select(e|e.isListClass());

Class getListElemTyp(Class c) :
 c.ownedAttribute.type.typeSelect(Class).first();

ActivityNode getInitNode(Activity this) :
 if(this.ownedElement.typeSelect
    (InitialNode).size > 0) then
  this.ownedElement.typeSelect(InitialNode).first
     ().outgoing.target.first()
 else
  this.ownedElement.typeSelect(ActivityNode).select
     (e|e.incoming.size == 0);

Boolean isInitialSend(Message msg) :
 let mos =
    (MessageOccurrenceSpecification)msg.sendEvent : (
  let frags = getLeadingFragments((
     (Interaction)msg.owner).fragment, mos) : (
   let prevMsg = getPrevMessageWithout2SelfRec
      (frags) : (
    (prevMsg != null) ? (
     let prevMos =
       (MessageOccurrenceSpecification)prevMsg.sendEvent :
         (mos.covered.first() != prevMos.covered.first())
    )
    :
     true
```

```
    )
   )
  )
;
Message getPrevMessageWithout2SelfRec
  (List[InteractionFragment] frags):
 (frags.isEmpty) ? null : (
  let frag = frags.last() : (
   (frag.metaType
     == MessageOccurrenceSpecification) ? (
    let mos = (MessageOccurrenceSpecification)frag : (
     (mos.message.isToSelf()) ? (
      getPrevMessageWithout2SelfRec(frags.withoutLast()))
     : ((Message)mos.message)))
   :
    getPrevMessageWithout2SelfRec(frags.withoutLast()))
 );

List[Message] getInitialSends
  (Interaction seq, Class termOrCard) :
 let lls = seq.ownedElement.typeSelect
   (Lifeline).select(l|l.represents.type.name
     == termOrCard.name) :
  getInitialSendsRec(seq.fragment, lls,
    (List[Message]){});
private cached List[Message] getInitialSendsRec
  (List[InteractionFragment] frags, List[Lifeline] lls, List[Message] msgs) :
 (frags.isEmpty) ?
  msgs
 : (
  let frag = frags.first() :
   (frag.metaType == MessageOccurrenceSpecification
    && ((MessageOccurrenceSpecification)frag).message.sendEvent
        == frag
    && ((MessageOccurrenceSpecification)frag).message.isInitialSend
        ()
    && !((MessageOccurrenceSpecification)frag).message.isToSelf()
    && !lls.intersect(frag.covered).isEmpty) ? (
    getInitialSendsRec(frags.withoutFirst(), lls,
      (List[Message])msgs.add((
        (MessageOccurrenceSpecification)frag).message)))
   : (getInitialSendsRec(frags.withoutFirst(), lls, msgs))
 );
Message getPrevMsg(Message msg) :
 let mos =
   (MessageOccurrenceSpecification)msg.sendEvent :
  let frags = (
    (Interaction)msg.owner).fragment.getLeadingFragments
      (mos) :
   getPrevMessageWithout2SelfRec(frags);
cached List[Message] getConsecutiveSends(Message msg) :
 let mos =
   (MessageOccurrenceSpecification)msg.sendEvent :
  let frags = (
```

```
    (Interaction)msg.owner).fragment.getFollowingFragments
      (mos) :
   getConsecutiveSendsRec(frags, mos.covered.first(),
     (List[Message]){});

List[Lifeline] getLifelines
  (Interaction seq, Class termcard) :
 seq.ownedElement.typeSelect(Lifeline).select
   (e|e.represents.type.name
     == termcard.name || e.represents.type.name
       == termcard.general.first().name);
private List[Message] getConsecutiveSendsRec
  (List[InteractionFragment] frags, Lifeline ll, List[Message] msgs) :
 let userLl = ll.owner.ownedElement.typeSelect
   (Lifeline).selectFirst(e|e.isUser()) : (
  if(frags.isEmpty) then
   msgs
  else (
   let frag = frags.first() :
    if(frag.metaType
      == MessageOccurrenceSpecification) then
     (let mos = (MessageOccurrenceSpecification)frag :
      if(mos.message.isToSelf()) then
       getConsecutiveSendsRec(frags.withoutFirst
         (), ll, msgs)
      else
       if(mos.message.sendEvent == mos &&
        (mos.covered.first()
          == ll || mos.covered.first() == userLl)
       ) then
        getConsecutiveSendsRec(frags.withoutFirst(),
        ll, (List[Message])msgs.add(mos.message))
       else
        if(((MessageOccurrenceSpecification)mos.message.sendEvent).
             covered.first() == ll
         || ((MessageOccurrenceSpecification)mos.message.sendEvent).
              covered.first() == userLl)
        then
         getConsecutiveSendsRec(frags.withoutFirst()
                                              , ll, msgs)
        else
         msgs
     )
    else
     getConsecutiveSendsRec(frags.withoutFirst
       (), ll, msgs)
  )
 );
cached List[InteractionFragment] getFollowingFragments
  (List[InteractionFragment] frags, MessageOccurrenceSpecification mos) :
 if(frags.first() == mos) then
  frags.withoutFirst()
 else
  getFollowingFragments(frags.withoutFirst(), mos);
```

```
cached List[InteractionFragment] getLeadingFragments
  (List[InteractionFragment] frags, MessageOccurrenceSpecification mos) :
 if(frags.last() == mos) then
  frags.withoutLast()
 else
  getLeadingFragments(frags.withoutLast(), mos);

Boolean isToSelf(Message msg) :
 (
   (MessageOccurrenceSpecification)msg.receiveEvent).covered
     == (
       (MessageOccurrenceSpecification)msg.sendEvent).covered;
essage getPrevReceivedMessage
  (Interaction seq, Message msg) :
 let mos =
   (MessageOccurrenceSpecification)msg.sendEvent :
  getPrevReceivedMessageRec
    (seq.fragment.getLeadingFragments(mos), mos);
private Message getPrevReceivedMessageRec
  (List[InteractionFragment] frags, InteractionFragment frag):
 if(frag.metaType
   == MessageOccurrenceSpecification) then
  (let mos = (MessageOccurrenceSpecification)frag :
   if(mos.message.isToSelf()) then
    getPrevReceivedMessageRec(frags.withoutLast
      (), frags.last())
   else
    if(mos.message.receiveEvent == mos) then
     mos.message
    else
     getPrevReceivedMessageRec(frags.withoutLast
       (), frags.last())
  )
 else
  getPrevReceivedMessageRec(frags.withoutLast
    (), frags.last());

List[Message] getNextMessagesToSelf(Message msg) :
 let seq = (Interaction)msg.owner :
  let mos =
    (MessageOccurrenceSpecification)msg.receiveEvent :
   let following = seq.fragment.getFollowingFragments
     (mos) :
    getNextMessagesToSelfRec(following.withoutFirst
      (), following.first(), (List[Message]){});
private List[Message] getNextMessagesToSelfRec
  (List[InteractionFragment] frags, InteractionFragment frag,
   List[Message] msgs) :
 if(frag.metaType
   == MessageOccurrenceSpecification) then
  (let mos = (MessageOccurrenceSpecification)frag :
   if(mos.message.isToSelf() && (
     (MessageOccurrenceSpecification)
       (mos.message.receiveEvent)) == mos) then
```

```
    getNextMessagesToSelfRec(frags.withoutFirst
      (), frags.first(), (List[Message])(msgs.add
        (mos.message)))
   else
    if(mos.message.receiveEvent == mos) then
     msgs
    else
     getNextMessagesToSelfRec(frags.withoutFirst
       (), frags.first(), msgs)
  )
 else
  getPrevMessagesToSelfRec(frags.withoutLast
    (), frags.last(), msgs);

List[Message] getPrevMessagesToSelf(Message msg) :
 let seq = (Interaction)msg.owner :
  let mos =
    (MessageOccurrenceSpecification)msg.sendEvent :
   getPrevMessagesToSelfRec
     (seq.fragment.getLeadingFragments(mos), mos,
       (List[Message]){}).reverse();
private List[Message] getPrevMessagesToSelfRec
  (List[InteractionFragment] frags, InteractionFragment frag,
   List[Message] msgs) :
 if(frag.metaType
   == MessageOccurrenceSpecification) then
  (let mos = (MessageOccurrenceSpecification)frag :
   if(mos.message.isToSelf() && (
     (MessageOccurrenceSpecification)
       (mos.message.receiveEvent)) == mos) then
    getPrevMessagesToSelfRec(frags.withoutLast
      (), frags.last(), (List[Message])(msgs.add
        (mos.message)))
   else
    if(mos.message.receiveEvent == mos) then
     msgs
    else
     getPrevMessagesToSelfRec(frags.withoutLast
       (), frags.last(), msgs)
  )
 else
  getPrevMessagesToSelfRec(frags.withoutLast
    (), frags.last(), msgs);

Parameter getFirstOperationInputParamater
  (Activity this):
 this.ownedElement.typeSelect
   (ActivityParameterNode).selectFirst
     (e|e.isInputParameterNode());

List[ActivityNode] getAllInitNodes(Class this) :
 this.getAllAcceptEventActions().union
   (this.getAllOperationsStartPoints());
```

```
List[AcceptEventAction] getAllAcceptEventActions
  (Class this):
 this.getModel().getAllActivities().reject
   (e|e.isOperation()).ownedElement.typeSelect
     (AcceptEventAction)
  .select(e|e.inPartition.first().getPartitionClass()
    ==this.name.toString());

List[ActivityNode] getAllOperationsStartPoints
  (Class this) :
 this.getModel().getAllActivities().select
   (e|e.ownedElement.typeSelect(ActivityNode).exists
     (e|e.isInputParameterNode
       ())).getFirstOperationInputParamater()
 .union(this.getModel().getAllActivities
   ().ownedElement.typeSelect(InitialNode)).typeSelect
     (ActivityNode)
 .select(e|e.inPartition.first().getPartitionClass()
   ==this.name.toString() || (
     (! this.superClass.isEmpty)
       ?(e.inPartition.first().getPartitionClass()
         ==this.superClass.first().name.toString
           ()):false));

List[Activity] getAllServiceOperations(Class this):
 this.getModel().getAllActivities().select
   (e|e.isServiceOperation());
List[StructuredActivityNode] getStructuredInitNodes
  (Class this) :
 this.getModel().getAllActivities
   ().ownedElement.typeSelect(StructuredActivityNode);
List[Class] getGeneralClasses
  (Classifier this) : this.general;

Class getGeneralClass(Classifier this):
 if(targetServer()) then
  getGeneralServer(this)
 else if(targetTerminal()) then
  getGeneralTerminal(this)
 else if(targetCard()) then
  getGeneralCard(this);

Class getGeneralCard(Classifier this) :
 this.getGeneralClasses().select(e|e.hasStereotype
   ("SecureMDD::Smartcard")).first() ;
Class getGeneralTerminal(Classifier this) :
 this.getGeneralClasses().select(e|e.hasStereotype
   ("SecureMDD::Terminal")
   ||e.hasStereotype("SecureMDD::PC")).first() ;

Class getGeneralServer(Classifier this) :
 this.getGeneralClasses().select(e|e.hasStereotype
   ("SecureMDD::Service")).first() ;
```

```
List[String] getUsedPredefinedInterfaces(Model mo) :
 (let res = { "Codeable" } :
 (let res1 = ( mo.requiresEncryption()
   ? res.add("PlainData") : res) :
 (let res2 = ( mo.requiresHashing()
   ? res.add("HashData") : res1) :
 (let res3 = ( mo.requiresSigning()
   ? res.add("SignData") : res2) :
 res3))));
List[String] getImplements
  (Class this) : this.getImplementedInterfaces
    ().collect(e|e.name);

Class getNumber(Model m) :
m.getSecurityDatatypes().ownedElement.typeSelect
   (Class).select(e|e.name.toString()
     =="Number").first();

Class getString(Model m) :
m.getSecurityDatatypes().ownedElement.typeSelect
   (Class).select(e|e.name.toString()
     =="String").first();

String getNameMessageSuperclass(Model m) :
m.getClassDiagram().ownedElement.typeSelect
   (Class).select(c|c.hasStereotype
     ("SecureMDD::Message")).select
       (c|c.getGeneralClasses().isEmpty).name.first();

List[Property] getAllPorts(Model m) :
m.getDeploymentDiagram().ownedElement.typeSelect
   (Node).ownedElement.typeSelect(Property);

List[Property] getTerminal2CardPorts(Model m) :
 getAllPorts(m).select(e|
  e.owner.hasStereotype("SecureMDD::Terminal")
  && e.type.hasStereotype("SecureMDD::Smartcard"));

List[Property] getCard2TerminalPorts(Model m) :
 getAllPorts(m).select(e|
  e.owner.hasStereotype("SecureMDD::Smartcard")
  && e.type.hasStereotype("SecureMDD::Terminal"));

String getGeneratingCard(Model m) :
 ((Class)getProperty
   (m,"generatingCard")).name.toString();

String getPackage(Class c) :
 c.isTerminal()
  ? ((
    (String)GLOBALVAR termpackage))+"."+c.name.toString
      ().toFirstLower()
  : (c.isCard()
   ? ((
```

```
      (String)GLOBALVAR cardpackage))+"."+c.name.toString
        ().toFirstLower()
   : "INVALID CLASS"
  );

String getDirectoryPrefix(Model m) :
 "/"+
 (targetCard()
  ?  ((Class)getProperty
     (m,"generatingCard")).name.toString().toFirstLower
       ()
  :  (targetTerminal()
     ? ((Class)getProperty
       (m,"generatingTerminal")).name.toString
         ().toFirstLower()
     : (targetTestcase()
       ? ((Class)getProperty
         (m,"generatingTestcase")).name.toString
           ().toFirstLower()
       : ((Class)getProperty
         (m,"generatingServer")).name.toString
           ().toFirstLower()
     )
   )
 )
 +"/";

Class getDirectoryPrefixClass(Model m):
 (targetCard()
  ?  ((Class)getProperty(m,"generatingCard"))
  :  (targetTerminal()
     ? ((Class)getProperty(m,"generatingTerminal"))
     : (targetTestcase()
       ? ((Class)getProperty(m,"generatingTestcase"))
       : ((Class)getProperty(m,"generatingServer"))
     )
   )
 );

String setDirectoryPrefix(Model m,Class generated) :
 (targetCard()
  ?  setProperty(m,"generatingCard",generated)
  :  (targetTerminal()
     ? setProperty(m,"generatingTerminal",generated)
     : (targetTestcase()
      ? setProperty(m,"generatingTestcase",generated)
      : setProperty(m,"generatingServer",generated)
     )
   )
 );

String getSendReceiveClass(String c) :
 if(c.startsWith("sendMsg") || c.startsWith
   ("byte")) then
```

```
  getSendClass(c)
 else
  if(c.startsWith("public") || c.startsWith
    ("private") ) then
   getReceiveClass(c)
  else
   "error_no_classname_found for " + c;

String getSendClass(String c) :
 if(targetCard()) then
  getSendClassRec(c.split(storeName() + ".new").get(1))
 else
  getSendClassRec(c.split("new ").get(1));

String getSeqSendClass(String c) :
 if(c.contains(":=")) then (
  let call = c.split(":=").get(1) :
   if(call.contains("(")) then
    call.split("\\(").first().trim()
   else
    call.trim()
 )
 else
  c.split("\\(").first().trim();

String getSendClassRec(String c):
 if(!c.endsWith(")")) then
  getSendClassRec(c.subString(0,c.length-1))
 else
  if(!c.subString(0,c.length-1).contains(")")) then
   c.subString(0,c.length-1)
  else
   getSendClassRec(c.subString(0,c.length-1))
  ;

String getSendLocVar(Message msg) :
 if(msg.name.contains(":=")) then
  msg.name.split(":=").get(0).trim()
 else
  ""
 ;

List[String] getSendClassAttrs(String s) :
 if(s.contains("(")) then (
  let args = s.split("\\(").get(1) :
    args.subString(0, args.length-1).split(",").trim()
  );

String getReceiveClass(String c):
 if(!c.startsWith("(")) then
  getReceiveClass(c.subString(1,c.length))
 else
  if(!c.endsWith(")")) then
   getReceiveClass(c.subString(0,c.length-1))
```

```
  else
   c.subString(1,c.length-" inmsg".length-1)
  ;

Class getClass(Model m,String s):
let c = (Class)m.getClassDiagramClasses().union
  (m.getSecurityDatatypesClasses()).selectFirst(e|(
    (Class)e).name==s):
if(c
  == null) then logging
    ("getClass: did not find " + s) else c;

Class getClass(String s, Model m):
m.getClass(s);

Class getMessageClass(Model mo, Message msg) :
 let msgName = msg.name.getSeqSendClass() :
  let seq = (Interaction)(msg.owner) :
   let gate = msg.getModel().getTestcaseClass
     (seq.name + gateName()) :
    if(mo.getClass(msgName) != null) then
     mo.getClass(msgName)
    else
     (Class)gate.ownedAttribute.typeSelect
       (Property).selectFirst(p|p.name == msgName).type
;

String getAttributeImplCode(Property p):
 (let s1=(
  if(p.visibility.toString()=="package") then
   ""
  else
   p.visibility.toString()):
  (let s2=(
   if(p.isLeaf) then
    s1+" final"
   else
    s1):
   (let s3=(
    if(p.isStatic) then
     s2+" static"
    else
     s2):
    (let s4=s3+" "+p.translateType
      ().translatePrimitiveType()+" "+p.name:
     if
       (p.defaultValue!=null && !p.defaultValue.stringValue
         ().matches
           ("") && p.type.metaType!=Enumeration) then
      (p.translateType()
        == "byte"
          ? s4+"= (byte) " +p.defaultValue.stringValue
            ()+";" : s4+"= " +p.defaultValue.stringValue
              ()+";")
```

```
      else
       s4+";"))));

String getStatic(Model mo, String p):
if((mo.getClassDiagramClasses().instanceOperations
  ().select(e|e.name==p).isStaticOp().first())
    ==true)then
 "static"
 else  ""
;
cached List[Class]getAllClassAndInterfaces(Model m):
 m.getClassDiagram().ownedElement.typeSelect(Class);
cached List[Node] getServersFromDeploymentDiagram
  (Model mo):
 mo.getDeploymentDiagram().ownedElement.typeSelect
   (Node).select(e|e.hasStereotype
     ("SecureMDD::Service"));
cached List[Node] getTerminalsFromDeploymentDiagram
  (Model mo):
 mo.getDeploymentDiagram().ownedElement.typeSelect
   (Node).select(e|e.hasStereotype
     ("SecureMDD::Terminal"));
cached Node getServerFromDeploymentDiagram
  (Model mo, String server):
  mo.getServersFromDeploymentDiagram().select
    (e|e.name==server).first();
cached Node getTerminalFromDeploymentDiagram
  (Model mo, String terminal):
  mo.getServersFromDeploymentDiagram().select
    (e|e.name==terminal).first();
cached Class getStatefulManager(Class server):
 let servs = server.getModel().getServers() :
 let du = logging("servers = " + servs.name) :
 servs.select(s|s.name
   == (server.name+"StatefulManager")).first();
cached Class getStatefulServer(Class serverManager):
 serverManager.getModel().getServers().select
   (s|s.name+"StatefulManager"
     ==serverManager.name).first();
cached Node getStatefulManager(Node server):
 server.getModel().getDeploymentDiagram
   ().ownedElement.typeSelect(Node).select(s|s.name
     ==server.name+"StatefulManager").first();
cached Node getStatefulServer(Node serverManager):
 serverManager.getModel().getDeploymentDiagram
   ().ownedElement.typeSelect(Node).select
     (s|s.name+"StatefulManager"
       ==serverManager.name).first();
cached Node getNode(Class c):
 c.getModel().getDeploymentDiagram
   ().ownedElement.typeSelect(Node).select(n|n.name
     ==c.name).first();
cached Node getNode(String s, Model mo):
 mo.getDeploymentDiagram().ownedElement.typeSelect
```

```
    (Node).select(n|n.name==s).first();
cached Node getNode(Type e):
 e.name.getNode(e.getModel());
cached Class getClass(Node n):
 n.getModel().getClassDiagramClasses().select
    (c|c.name==n.name).first();
cached Node getPortType(Model mo, String portName):
 (Node)mo.getDeploymentDiagram
    ().ownedElement.typeSelect
      (Node).ownedElement.typeSelect
        (Property).selectFirst(e|e.name==portName).type;
cached List[Node] getDirectlyUsedServers
  (Class terminalServer):
 let outgoingNodes=getOutgoingNodes
    (terminalServer.name,terminalServer.getModel()):
 terminalServer.getModel().getDeploymentDiagram
    ().ownedElement.typeSelect(Node).select
      (n|outgoingNodes.exists(u|n.hasStereotype
        ("SecureMDD::Service") && (u.name
          ==n.name || n.name+"StatefulManager"
            ==terminalServer.name)));
cached List[Node] getInvokerNodes
  (String nodeName,Model mo):
 getIncommingAssociations
    (mo,nodeName).memberEnd.type.typeSelect
      (Node).select(n| n.name!=nodeName).removeDoubles
        ();
cached List[Association] getIncommingAssociations
  (Model mo,String nodeName):
 let node= getNode(nodeName,mo):
 node.getAssociations().select(a|a.memberEnd.exists
    (e| e.type==node && e.isNavigable()) );
cached List[Node] getOutgoingNodes
  (String nodeName,Model mo):
 let node= getNode(nodeName,mo):
 node.attribute.type.typeSelect(Node);
cached List[Node] getOutgoingNodes(Node node):
 node.attribute.type.typeSelect(Node);
cached List[String] getServerNames(Model mo):
 let servers= mo.getServersFromDeploymentDiagram
    ().name:
 let blub=mo.getServersFromDeploymentDiagram().collect
    (e|
  if(e.isStateful()) then{
   servers.add(e.name+"StatefulManager")
  }) :servers.sortBy(e|e);

String getPortFromTo(Class from, Class to) :
 from.getModel().getAllPorts().selectFirst
    (p|p.class.name == from.name && p.type.name
      == to.name).name;
cached List[Constraint] getConstraints(Element el) :
 el.getModel().getAllTestcases
    ().ownedElement.typeSelect(Constraint).select
```

```
      (c|c.constrainedElement.contains(el));
cached List[String] getConstraintBodies(Element el) :
 el.getConstraints().getConstraintBodies();

List[String] getConstraintBodies(Constraint constr) :
 constr.eAllContents.typeSelect
   (OpaqueExpression).first().body;

Property getRedefininingProperty
  (Class cla, Property redefProperty) :
 cla.ownedAttribute.selectFirst
   (a|a.redefinedProperty.exists(r|
  r.name == redefProperty.name
  && r.type.name == redefProperty.type.name
  && r.class.name == redefProperty.class.name
  ));
cached Class getUserClass(Model mo):
 mo.getClassDiagramClasses().
 selectFirst(e|(e.hasStereotype("SecureMDD::User")));
cached Class getDefaultGateClass(Model mo):
 mo.getTestcaseClasses().select(e|e.name
   =="DefaultGate").first();
cached List[Class] getGateClasses(Model mo):
 mo.getTestcaseClasses().select(e|e.name.endsWith
   (gateName())).reject(e|e
     == mo.getDefaultGateClass());
String getGateClassname(String name) :
 name.toFirstUpper() + "Gate";
cached Interface getConstraintInterface(Model mo):
 mo.getTestcaseInterfaces().select(e|e.name
   ==constraintName()).first();
cached Interface getGateInterface(Model mo):
 mo.getTestcaseInterfaces().select(e|e.name
   ==gateName()).first();
cached String getNamespace(String server):
 let package = ((String)GLOBALVAR package):
  let rpath= package.split("\\.").reverse().toList():
   "http://"+server.toFirstLower()+"."+rpath.get
     (0)+"."+rpath.get(1)+"."+rpath.get(2)+"/";
cached String getNamespace
  (String server, String componentType):
 let package = ((String)GLOBALVAR package):
  let rpath= package.split("\\.").reverse().toList():
   "http://"+server.toFirstLower
     ()+"."+componentType+"."+rpath.get
       (1)+"."+rpath.get(2)+"/";
cached Property getServiceIdentifier(Class c):
 c.getAllAttributesG().selectFirst(e|e.hasStereotype
   ("SecureMDD::Identifier"));
cached String getEmptyDefaultReturnType(Parameter p):
 if p == null then
  ""
 else if p.translateType()=="boolean" then
  "true"
```

```
 else if p.translateType()
   =="int" || p.translateType()=="short" then
  "0"
 else
  "null";
```

3.3.4 Operations which define names

```
import uml;
extension psm_templates_shared::tests;
extension org::eclipse::xtend::util::stdlib::elementprops;
extension psm_templates_shared::getter;
cached String successName() : "SUCCESS";
cached String abortName() : "stop";
cached String unhandledAbortName() : "raiseError";
cached String abortClassName() : simpleCommName() +
                                  "." + abortName();
cached String codingName() : "Coding";
cached String codeableName() : "Codeable";
cached String codeName() : "Code";
cached String constantsName() : "Constants";
cached String secretName() : "Secret";
cached String nonceName() : "Nonce";
cached String keyName() : "Key";
cached String symmkeyName() : "SymmKey";
cached String asymmkeyName() : "Asymmkey";
cached String pubkeyName() : "PublicKey";
cached String privkeyName() : "PrivateKey";
cached String keypairName() : "KeyPair";
cached String utilsName() : "Utils";
cached String hashDataName() : "HashData";
cached String hashedDataName() : "HashedData";
cached String plainDataName() : "PlainData";
cached String encDataName() : "EncData";
cached String encDataSymmName() : "EncDataSymm";
cached String encDataAsymmName() : "EncDataAsymm";
cached String signDataName() : "SignData";
cached String signedDataName() : "SignedData";
cached String macDataName() : "MACData";
cached String simpleCommName() : "SimpleComm";
cached String swtReaderName() : "SWTReader";
cached String smartCardSimulationName()
                    : "SmartCardSimulation";
cached String readerName() : "Reader";
cached String simReaderName() : "SimReader";
cached String arraysName() : "Arrays";
cached String mathName() : "Math";
cached String storeName() : "Store";
cached String userinterfaceName() : "Userinterface";
cached String enumTypeName() : "byte";

String packagenameCard(Model m) :
 (((String)GLOBALVAR package))+"."+(
   (Class)m.getProperty
```

```
      ("generatingCard")).name.toString().toFirstLower();

String packagenameTerminal(Model m) :
 (((String)GLOBALVAR package))+"."+(
   (Class)m.getProperty
     ("generatingTerminal")).name.toString
       ().toFirstLower();

String packagenameServer(Model m) :
 (((String)GLOBALVAR package))+"."+(
   (Class)m.getProperty
     ("generatingServer")).name.toString
       ().toFirstLower();

String packagenameTestcase(Model m) :
 (((String)GLOBALVAR package));

String packagename(Model m) :
 if(targetServer()) then
  packagenameServer(m)
 else if(targetTerminal()) then
  packagenameTerminal(m)
 else if(targetCard()) then
  packagenameCard(m)
 else "Error: Target is not Server, Terminal or Card";

String packagename() :
 if(targetServer()) then
  "packagenameServer()"
 else if(targetTerminal()) then
  "packagenameTerminal()"
 else if(targetCard()) then
  "packagenameCard()"
 else "Error: Target is not Server, Terminal or Card";

String packageToPathName(String package) :
 package.replaceAll("\\.","/");

String manualFileName(Class c):
 c.getModel().getDirectoryPrefixClass
   ().name.toFirstLower()+"/"+"Manual_"+c.name+".java";

String manualClassName(Class c):
 c.getModel().packagenameManual()+"."+"Manual_"+c.name;

String packagenameManual(Model m) :
 ((String)GLOBALVAR package_manual)+"."+m.getDirectoryPrefixClass
      ().name.toFirstLower();

String projectName(Model m):
 m.name.toFirstLower();
```

```
String componentType(Model m):
 if(targetServer()) then
  packagenameServer(m)
 else if(targetTerminal()) then
  packagenameTerminal(m)
 else if(targetCard()) then
  packagenameCard(m)
 else "Error: Target is not Server, Terminal or Card";

String filename(NamedElement this) :
 this.name.toString()+".java";

String getPartitionClass(ActivityPartition this) :
 let n = this.name :
 ((n != null && n.length > 0) ?
  (n.split(":") != null && n.split
    (":").size > 1 && n.split(":").get(1) != null ?
   n.split(":").get(1).trim() : n.trim())
 : ((Class)this.represents).name)
 ;
String lengthConstantsName() : "LengthConstants";
String copyName() : "copy";
String terminalExceptionName() : "TerminalException";
String serverExceptionName() : "ServiceException";
String cardExceptionName() : "ISOException";
String exceptionName() :
 if(targetServer()) then
  serverExceptionName()
 else if(targetTerminal()) then
  terminalExceptionName()
 else if(targetCard()) then
  cardExceptionName()
 else "Error: Target is not Server, Terminal or Card";
String deploymentPropertiesName() : "Ports";
String messageWrapperName(): "MessageWrapper";
String stubsGeneratorName(): "StubsGenerator";
String userName(): "User";
String testcaseName() : "Testcase";
String gateName() : "Gate";
String constraintName() : "Constraint";
String runAllTestsName() : "AllTests";
String defaultSecret() : "new byte[]{..}";
String defaultSymmkey() : "new byte[]{..}";
String defaultPubModulus() : "new byte[]{..}";
String defaultPrivExponent() : "new byte[]{..}";
```

3.3.5 Transformations that generate several classes for smart cards and terminals

The transformations described in this subsection generate the class `SimpleComm` for the smart cards as well as other classes, e.g. the security datatypes, the class `Math` and the class `Array`.

```
≪IMPORT uml≫
≪EXTENSION psm_templates_shared::getter≫
≪EXTENSION psm_templates_shared::names≫
```

```
«EXTENSION psm_templates_shared::tests»
«EXTENSION psm_templates_shared::parse»
«EXTENSION psm_templates_shared::translations»

«DEFINE generateSimplecommClasses FOR Class»
«LET this.getModel() AS mo»
«FOREACH this.getModel()
  .getAllClassAndInterfaces() AS c»
 «IF c.name==codeName()»
  «EXPAND generateCodeClass FOR c»
 «ENDIF»
 «IF c.name==mathName()»
  «EXPAND generateMath FOR c»
 «ENDIF»
 «IF c.name==simpleCommName()»
  «EXPAND generateSimpleComm(this) FOR c»
 «ENDIF»
 «IF c.getImplementedInterfaces().toString()
   .contains(codeableName())»
   «EXPAND generateCodeableClass FOR this»
 «ENDIF»
 «IF c.name==arraysName()»
   «EXPAND generateArraysClass FOR c»
 «ENDIF»
 «IF c.ownedAttribute.union(c.ownedOperation
   .ownedParameter).exists(e|e.type
     .name==secretName() || mo
       .requiresSymmetricEncryption())»
  «EXPAND generateSecretClass FOR mo»
 «ENDIF»
 «IF c.ownedAttribute.union(c.ownedOperation
   .ownedParameter).exists(e|e.type
     .name==nonceName())»
  «EXPAND generateNonceClass FOR mo»
 «ENDIF»
 «IF c.name.startsWith("ListOf")»
  «EXPAND generateListClass FOREACH mo
    .getListClasses()»
 «ENDIF»
  «IF c.name==constantsName()»
  «EXPAND generateConstantsClass FOREACH mo
    .getAllConstants()»
 «ENDIF»
«ENDFOREACH»
 «IF mo.requiresHashing()||mo.requiresMAC()»
  «EXPAND generateHashDataClass FOR mo»
  «EXPAND generateHashedDataClass FOR mo»
 «ENDIF»
 «IF mo.requiresEncryption()||mo.requiresMAC()»
  «EXPAND generatePlainDataClass FOR mo»
  «EXPAND generateEncDataClass FOR mo»
  «EXPAND generateKeyClass FOR mo»
 «ENDIF»
 «IF mo.requiresSymmetricEncryption()||mo
```

```
    .requiresMAC()||mo.requiresSymmKey()≫
  ≪EXPAND generateSymmKeyClass FOR mo≫
  ≪EXPAND generateUtilsClass FOR mo≫
 ≪ENDIF≫
 ≪IF mo.requiresSigning()-≫
  ≪EXPAND generateKeyClass FOR mo≫
  ≪EXPAND generateSignDataClass FOR mo≫
  ≪EXPAND generateSignedDataClass FOR mo≫
 ≪ENDIF≫
 ≪IF mo.requiresSymmetricEncryption()||mo
    .requiresMAC()≫
  ≪EXPAND generateEncDataSymmClass FOR mo≫
 ≪ENDIF≫
 ≪IF mo.requiresAsymmetricEncryption()≫
  ≪EXPAND generateEncDataAsymmClass FOR mo≫
 ≪ENDIF≫
 ≪IF mo.requiresPrivateKey()||mo
    .requiresAsymmetricEncryption()||mo
      .requiresSigning()≫
  ≪EXPAND generatePrivateKeyClass FOR mo≫
 ≪ENDIF≫
 ≪IF mo.requiresPublicKey()||mo
    .requiresAsymmetricEncryption()||mo
      .requiresSigning()≫
  ≪EXPAND generatePublicKeyClass FOR mo≫
 ≪ENDIF≫
 ≪IF (mo.requiresPublicKey()||mo
    .requiresPublicKey()) && !this.isCard()≫
  ≪EXPAND generateKeyPairClass FOR mo≫
 ≪ENDIF≫
 ≪IF mo.requiresMAC()≫
  ≪EXPAND generateMACDataClass FOR mo≫
 ≪ENDIF≫
 ≪EXPAND generateEnumClasses FOREACH mo
    .getAllEnumerations()≫
≪ENDLET≫
≪ENDDEFINE≫

≪DEFINE generateArraysClass FOR Class≫
≪LET this.getModel() AS mo≫
≪FILE mo.getDirectoryPrefix()+arraysName()
  +".java"-≫
package ≪mo.packagename()-≫;
≪IF targetCard()≫
import javacard.framework
  .Util;≪ELSE≫import swt.util.ByteArray;
≪ENDIF≫
public class ≪arraysName()≫ {
public static boolean equals
  (boolean[] left, boolean[] right) {
 if (left.length != right.length) return false;
 for (short i = 0; i < left.length; i+
   +) if (left[i] != right[i]) return false;
 return true;
```

```
}
≪IF targetServer()≫
public static boolean equals
  (String left, String right) {
 return left.equals(right);
}
≪ENDIF≫
public static boolean equals
  (byte[] left, byte[] right) {
 if (left.length != right.length) return false;
 ≪IF targetServer()||targetTerminal
   ()≫return ByteArray.equals(left, right);
 ≪ENDIF≫
 ≪IF targetCard()≫
 return (Util.arrayCompare(left, (short) 0,
     right, (short) 0, (short) left.length) == 0);
 ≪ENDIF≫
}
public static boolean equals
  (short[] left, short[] right) {
 if (left.length != right.length) return false;
 for (short i = 0; i < left.length; i+
   +) if (left[i] != right[i]) return false;
 return true;
}
≪IF !targetCard()≫
public static boolean equals(≪codeableName
  ()≫[] left, ≪codeableName()≫[] right) {
 if (left.length != right.length)
  return false;
 for (short i = 0; i < left.length; i++) {
  if (!(left[i].equals(right[i])))
   return false;
 }
 return true;
}
≪ENDIF≫
≪IF targetCard()≫
public static void copy(byte[] from, byte[] to) {
 Util
   .arrayCopy(from, (short)0, to, (short) 0,
     (short)from.length);
}
public static void copy
  (byte[] from, short offset, short length
    , byte[] to) {
 Util
   .arrayCopy(from, offset, to, (short) 0,
     (short) length);
}
≪ENDIF≫
}
≪ENDFILE≫
≪ENDLET≫
```

```
«ENDDEFINE»

«DEFINE generateListClass FOR Class»
«FILE this.getModel().getDirectoryPrefix()
  +filename()-»
«LET this.ownedAttribute.type.typeSelect(Class)
  .first() AS c»
package «this.getModel().packagename()-»;
«IF targetCard()»
import javacard.framework.ISOException;
import javacard.framework.ISO7816;
«ELSE»
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import java.util.ArrayList;
@XmlAccessorType(XmlAccessType.FIELD)
«ENDIF»
public class «this.name» «this.getModel()
  .ppImplements()»{
«IF !targetCard()&&this.ownedAttribute.first()
  .upper!=-1||targetCard()»
 private short size;
 «IF targetCard()»public «this
   .getListElemTyp()
     .name»[] elems; «ELSE»public «this
       .ownedAttribute.type.typeSelect(Class).first()
         .name»[] elems;«ENDIF»
 «IF targetCard
   ()»public«ELSE»private«ENDIF» short index;
 «IF targetTerminal() || targetServer()»
   public «this.name»(){
  this.size = 0;
   elems = new «this.getListElemTyp()
     .name»[size];
   for (short i = 0; i < elems.length; i++)
    elems[i] = new «this.getListElemTyp()
      .name»();
  }
 «ENDIF»
 public «this.name»(short size) {
  this.size = size;
  elems = new «IF !targetCard()» «c
    .name»[size]; «ELSE» «this
      .getListElemTyp().name»[size];«ENDIF»
  for (short i = 0; i < elems.length; i++)
   elems[i] = new«IF !targetCard()» «c
     .name»();«ELSE» «this
       .getListElemTyp().name»();«ENDIF»
 }
 public void add(«IF !targetCard()» «c
   .name»«ELSE» «this.getListElemTyp()
     .name»«ENDIF» e) «IF targetTerminal
       () || targetServer()»
         throws «exceptionName
```

```
            ()≫≪ENDIF≫{
  if(hasFree()) {
   elems[index++].copy(e);
  }
  else {
   stop();
  }
 }
 public short size() {
 return index;
 }
 public boolean hasFree() {
 return index < elems.length;
 }
 public short index(){
 return index;
 }
 /
 public class ≪name–≫ {
  ≪FOREACH attribute AS attr–≫
   public ≪IF this
     .hasStaticAttributes
       ()–≫static ≪ENDIF–≫final short ≪attr
         .name–≫=
   ≪IF (attr.getDefault()
     .toString() == "null") || attr.getDefault()
       .toString().length==0–≫
    ≪attribute.indexOf(attr)+1–≫
   ≪ELSE–≫
    ≪attr.getDefault()–≫
   ≪ENDIF–≫;
  ≪ENDFOREACH–≫
 }
≪ENDFILE≫
≪ENDDEFINE≫
/
   ≪ELSE≫
≪ENDIF≫
   public interface ≪hashDataName
     ()≫ extends ≪codeableName()≫ {}
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateSecretClass FOR Model≫
≪FILE this.getModel().getDirectoryPrefix()
  +secretName()+".java"–≫
package ≪this.packagename()–≫;
≪IF targetCard()≫
import javacard.framework.Util;
≪ELSE≫
import swt.util.ByteArray;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
@XmlAccessorType(XmlAccessType.FIELD)≪ENDIF≫
```

```
public class ≪secretName()≫ ≪this
  .ppImplements()≫ {
≪IF targetServer()≫
private byte[] secret;
 ≪ENDIF≫
 ≪IF targetCard()||targetTerminal()≫
 ≪IF targetTerminal()≫
 private≪ELSE≫public≪ENDIF≫ byte[] secret;
private static final byte[] defaultSecret = ≪defaultSecret
  ()≫; ≪ENDIF≫
public Secret() {
≪IF targetTerminal()≫
secret = defaultSecret;
≪ENDIF≫
≪IF targetCard()≫
secret = new byte[Store.LENGTHOFSECRET];
 Util
   .arrayCopy(defaultSecret, (short)0, secret,
     (short)0, (short)(defaultSecret.length > Store
       .LENGTHOFSECRET ? Store
         .LENGTHOFSECRET : defaultSecret.length));
≪ENDIF≫
}
public Secret(byte[] secret) {
 this.secret = secret;
}
≪IF !targetCard()≫
public Secret(String secret){
 this.secret=java.util.Arrays.copyOfRange(secret
   .getBytes(),0,LengthConstants.LENGTHOFSECRET);
}
≪ENDIF≫
 ≪IF targetServer()||targetTerminal()≫
 public byte[] getSecret() {
 return secret;
}
public void setSecret(byte[] secret) {
 this.secret=secret.clone();
}≪ENDIF≫
public byte getCode() {
 return Code.SECRET;
}
public boolean equals(Secret other) {
 return Arrays.equals(secret, other.secret);
}
public void copy(Secret from) {
 ≪IF targetCard()≫
   Arrays.copy(from.secret, this.secret);
 ≪ELSE≫this.secret= from.secret
   .clone();≪ENDIF≫
}
}
≪ENDFILE≫
≪ENDDEFINE≫
```

```
≪DEFINE generateNonceClass FOR Model≫
≪FILE this.getModel().getDirectoryPrefix()
  +nonceName()+".java"−≫
 package ≪this.packagename()−≫;
 ≪IF targetCard()≫
 import javacard.security.RandomData;
 import javacard.framework.*;
 ≪ELSE≫
 import java.security.SecureRandom;
 import swt.util.ByteArray;
 import javax.xml.bind.annotation.XmlAccessType;
 import javax.xml.bind.annotation.XmlAccessorType;
 import javax.xml.bind.annotation.XmlTransient;
 @XmlAccessorType(XmlAccessType.FIELD)
 ≪ENDIF≫
 public class Nonce ≪this.ppImplements()≫ {
 ≪IF targetCard()≫
 public≪ELSE≫private≪ENDIF≫ byte[] nonce;
 ≪IF targetCard()≫
 private static RandomData rd;
 ≪ELSE≫
 @XmlTransient
 private SecureRandom rd;≪ENDIF≫
public Nonce() {
≪IF targetCard()≫
if (rd == null) rd = RandomData.getInstance(RandomData
   .ALG_SECURE_RANDOM);
    nonce = new byte[Store.LENGTHOFNONCE];
    rd.generateData(nonce, (short) 0, (short) nonce
      .length);
≪ELSE≫
 nonce = new byte[≪lengthConstantsName()≫
    .LENGTHOFNONCE];
 rd= new SecureRandom();≪ENDIF≫
}
≪IF targetServer()||targetTerminal()≫
public byte[] getNonce() {
 return nonce;
}
public void setNonce(byte[] nonce) {
 ByteArray.copy(nonce, (short) 0, this
    .nonce, (short) 0,
    (short) this.nonce.length);
}
≪ELSE≫
public Nonce(byte[] nonce) {
    this.nonce = nonce;
    }
≪ENDIF≫
public void copy(Nonce o) {
≪IF targetCard()≫
Arrays.copy(o.nonce, nonce);
≪ELSE≫
 nonce=o.getNonce().clone();≪ENDIF≫
```

```
}
public byte getCode() {
 return Code.NONCE;
}
public boolean equals(Nonce other) {
 return Arrays.equals(nonce, other.nonce);
}
public static Nonce generateNonce() {
≪IF targetCard()≫
if (rd == null) rd = RandomData.getInstance(RandomData
  .ALG_SECURE_RANDOM);
 return Store.newNonce().instGenerateNonce();
 ≪ELSE≫return (new Nonce())
   .instGenerateNonce();≪ENDIF≫
}
public Nonce instGenerateNonce() {
 ≪IF targetServer()||targetTerminal()≫
 rd.nextBytes(nonce);≪ELSE≫
 rd.generateData(nonce, (short) 0, (short)nonce
   .length);
 ≪ENDIF≫
 return this;
}
}
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateKeyClass FOR Model≫
≪FILE this.getModel().getDirectoryPrefix()
  +keyName()+".java"−≫
package ≪this.packagename()−≫;
public abstract class Key {
public boolean isSymmKey() {
 return false;
}
public boolean isPublicKey() {
 return false;
}
public boolean isPrivateKey() {
 return false;
}
≪IF targetCard()≫
public abstract javacard.security.Key toAPIKey() ;
≪ENDIF≫
}
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateSymmKeyClass FOR Model≫
≪FILE this.getModel().getDirectoryPrefix()
  +symmkeyName()+".java"−≫
package ≪this.packagename()−≫;
≪IF targetCard()≫
import javacard.security.DESKey;
```

```
import javacard.security.KeyBuilder;
import javacard.security.RandomData;
≪ELSE≫
import java.security.spec.*;
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.DESedeKeySpec;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlTransient;
@XmlAccessorType(XmlAccessType.FIELD)≪ENDIF≫
public class SymmKey extends Key ≪this
  .ppImplements()≫ {
public final static short SYMMKEYLENGTH = 24;
≪IF targetServer()||targetTerminal()≫
public final static String cryptProvider = "BC";
@XmlTransient
private SecureRandom rand;≪ENDIF≫
≪IF targetTerminal()||targetCard()≫
private final static byte[] defaultSymmKey = ≪defaultSymmkey
  ()≫;≪ENDIF≫
≪IF targetCard()≫
private DESKey apikey;
private static RandomData rd;≪ENDIF≫
≪IF targetCard()≫
public≪ELSE≫private ≪ENDIF≫ byte[] key;
public SymmKey() {
≪IF targetServer()||targetTerminal()≫
 Security.addProvider(new org.bouncycastle.jce
   .provider.BouncyCastleProvider());
 rand = new SecureRandom();
≪ENDIF≫
 ≪IF targetTerminal()≫
 key=defaultSymmKey;
 ≪ENDIF≫
 ≪IF targetCard()≫
 key = new byte[SYMMKEYLENGTH];
 apikey = (DESKey)KeyBuilder.buildKey(KeyBuilder
   .TYPE_DES,
        KeyBuilder.LENGTH_DES3_3KEY,
        false);
 Arrays.copy(defaultSymmKey, this.key);
 updateKey();
 ≪ENDIF≫
}
 public SymmKey≪IF targetServer()||targetTerminal
   ()≫(byte[] k) ≪ELSE≫
     (byte[] key)≪ENDIF≫{
 this();
 ≪IF targetCard()≫
 Arrays.copy(key, this.key);
 updateKey();
 ≪ELSE≫setKey(k);≪ENDIF≫
}
```

```
public static SymmKey generateKey() {
 ≪IF targetCard()≫
 if (rd == null) rd = RandomData
   .getInstance(RandomData.ALG_SECURE_RANDOM);
 return Store.newSymmKey()
   .instGenerateKey();≪ELSE≫
 return (new SymmKey()).instGenerateKey();
 ≪ENDIF≫
}
public static SymmKey generateKeyFromSecret(Secret s) {
  ≪IF targetCard()≫
  return Utils.instGenerateKey(Store
    .newSymmKey(),s);≪ELSE≫
  return Utils.instGenerateKey(new SymmKey(),s);
  ≪ENDIF≫
 }
public SymmKey instGenerateKey() {
 ≪IF targetCard()≫
 rd.generateData(key, (short) 0, (short) SymmKey
   .SYMMKEYLENGTH);
 updateKey();
 ≪ELSE≫
 byte[] k = new byte[SYMMKEYLENGTH];
 rand.nextBytes(k);
 setKey(k);≪ENDIF≫
 return this;
}
public boolean equals(SymmKey other) {
 return Arrays.equals(key, other.key);
}
public byte getCode() {
 return Code.SYMMKEY;
}
≪IF targetCard()≫
public void updateKey() {
 apikey.setKey(key, (short) 0);
}
≪ELSE≫
public void setKey(byte[] k) {
 key=k.clone();
}
≪ENDIF≫
public void copy(SymmKey s) {
 ≪IF targetCard()≫
Arrays.copy(s.key, key);
 apikey.setKey(key, (short)0);
≪ELSE≫ if (s.getKey() != null) {
  setKey(s.key);
 } ≪ENDIF≫
}
public boolean isSymmKey() {
 return true;
}
≪IF targetCard()≫
```

```
public javacard.security.Key toAPIKey() {
 return apikey;
}
≪ELSE≫
public SecretKey toAPIKey() throws Exception{
 try{
  SecretKeyFactory kf = SecretKeyFactory
    .getInstance("DESede",
    cryptProvider);
  SecretKey apikey = kf
    .generateSecret(new DESedeKeySpec(key));
  return apikey;}
  catch(Exception e){
   return null;
  }
}
public byte[] getKey() {
 return key;
}≪ENDIF≫
}
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateUtilsClass FOR Model≫
≪FILE this.getModel().getDirectoryPrefix()
  +utilsName()+".java"−≫
package ≪this.packagename()−≫;
≪IF targetCard()−≫
import javacard.security.MessageDigest;
import javacard.framework.Util;
≪ELSE−≫
import swt.util.ByteArray;
import java.security.spec.*;
import java.security.*;
≪ENDIF−≫
public class Utils {
 ≪IF targetCard()≫
 private static byte[] secret_long = new byte[9];
 private static byte[] hashvalue = new byte[
   (short) 20];
 private static MessageDigest md;
 public static SymmKey instGenerateKey
   (SymmKey sk, Secret s) {
  if (md == null)
  md = MessageDigest.getInstance(MessageDigest
    .ALG_SHA, false);
  Arrays.copy(s
    .secret, (short) 0, (short) 8, secret_long);
  secret_long[8] = (byte) 1;
  md
    .doFinal(secret_long, (short) 0,
      (short) secret_long.length,
    hashvalue, (short) 0);
  Arrays.copy(hashvalue, (short) 0, (short) 20, sk
```

```
    .key);
  secret_long[8] = (byte) 2;
  md
    .doFinal(secret_long, (short) 0,
      (short) secret_long.length,
    hashvalue, (short) 0);
  Util.arrayCopy(hashvalue, (short) 0, sk
    .key, (short) 20, (short) 4);
  sk.updateKey();
  return sk;
 }≪ENDIF≫
 ≪IF !targetCard()-≫
 private static byte[] hashvalue = new byte[20];
 private static byte[] secret_long = new byte[9];
 private static MessageDigest md;
 public final static String cryptProvider = "BC";
 public static SymmKey instGenerateKey
   (SymmKey sk, Secret s) {
  Security.addProvider(new org.bouncycastle.jce
    .provider.BouncyCastleProvider());
  byte[] k = new byte[SymmKey.SYMMKEYLENGTH];
  try{
   md = MessageDigest
     .getInstance("SHA1", cryptProvider);
   ByteArray.copy(s.getSecret(), secret_long, 0);
   secret_long[8] = (byte) 1;
   hashvalue = md.digest(secret_long);
   ByteArray.copy(hashvalue, 0, k, 0, 20);
   secret_long[8] = (byte) 2;
   hashvalue = md.digest(secret_long);
   ByteArray.copy(hashvalue, 0, k, 20, 4);
   sk.setKey(k);
   return sk;
  }
  catch(Exception e){
   return null;
  }
 }
 ≪ENDIF-≫
}
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateKeyPairClass FOR Model≫
≪FILE this.getModel().getDirectoryPrefix()
  +keypairName()+".java"-≫
package ≪this.packagename()-≫;
import java.math.BigInteger;
import java.security.KeyPairGenerator;
import java.security.SecureRandom;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import swt.util.ByteArray;
public class KeyPair {
```

```
 private final static String cryptProvider = "BC";
 private static void initJCE() {
  java.security.Security.addProvider(new org
    .bouncycastle.jce.provider.BouncyCastleProvider());
 }
 public PublicKey publicKey;
 public PrivateKey privateKey;
 public PublicKey getPublicKey() { return publicKey; }
 public PrivateKey getPrivateKey
   () { return privateKey; }
 public KeyPair
   (PublicKey publicKey, PrivateKey privateKey){
  this.publicKey = publicKey;
  this.privateKey = privateKey;
 }
 public static KeyPair generateKeyPair()
   throws Exception {
  initJCE();
  SecureRandom rand = new SecureRandom();
  java.security.KeyPair kp = null;
  KeyPairGenerator keyGen = KeyPairGenerator
    .getInstance("RSA", cryptProvider);
  keyGen.initialize(1024,rand);
  kp = keyGen.generateKeyPair();
  RSAPrivateKey priv = (RSAPrivateKey)kp.getPrivate();
  RSAPublicKey pub = (RSAPublicKey)kp.getPublic();
  BigInteger mod_bi = priv.getModulus();
  BigInteger pre_bi = priv.getPrivateExponent();
  BigInteger pue_bi = pub.getPublicExponent();
  byte[] pub_exp = pue_bi.toByteArray();
  byte[] priv_exp = pre_bi.toByteArray();
  if (priv_exp.length == 129) priv_exp = ByteArray
    .subarray(priv_exp, 1);
  byte[] modulus = mod_bi.toByteArray();
  if (modulus.length == 129) modulus = ByteArray
    .subarray(modulus, 1);
  PublicKey pubkey = new PublicKey(modulus);
  PrivateKey privkey = new PrivateKey
    (modulus, priv_exp);
  return new KeyPair(pubkey, privkey);
 }
}
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generatePublicKeyClass FOR Model≫
≪FILE this.getModel().getDirectoryPrefix()
  +pubkeyName()+".java"−≫
package ≪this.packagename()−≫;
≪IF targetCard()≫
import javacard.security.RSAPublicKey;
import javacard.security.KeyBuilder;
≪ELSE≫
import java.math.BigInteger;
```

```
import java.security.spec.InvalidKeySpecException;
import java.security.*;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
@XmlAccessorType(XmlAccessType.FIELD)
≪ENDIF≫
public class PublicKey extends Key ≪this
  .ppImplements()≫ {
public final static short MODULUSLENGTH = 128;
public final static short EXPONENTLENGTH = 3;
≪IF targetCard()≫
 private RSAPublicKey apikey;
≪ELSE≫public final static String cryptProvider = "BC";≪ENDIF≫
≪IF targetServer()||targetTerminal()≫
private≪ELSE≫public≪ENDIF≫ byte[] modulus;
≪IF targetServer()||targetTerminal()≫
private≪ELSE≫public≪ENDIF≫ byte[] pubexponent = {0x01
  , 0x00, 0x01};
private static final byte[] defaultModulus = ≪defaultPubModulus
  ()≫;
public PublicKey() {
≪IF targetTerminal()||targetServer()≫
  Security.addProvider(new org.bouncycastle.jce
    .provider.BouncyCastleProvider());
  setKey(defaultModulus);
≪ENDIF≫
≪IF targetCard()≫
modulus = new byte[MODULUSLENGTH];
 apikey = (RSAPublicKey)KeyBuilder.buildKey(KeyBuilder
   .TYPE_RSA_PUBLIC,
        KeyBuilder.LENGTH_RSA_1024,
        false);
 Arrays.copy(defaultModulus, this.modulus);
 updateKey();≪ENDIF≫
}
public PublicKey(byte[] modulus){
 this();
 ≪IF targetCard()≫
 Arrays.copy(modulus, this.modulus);
 updateKey();≪ELSE≫
 setKey(modulus);
 ≪ENDIF≫
}
public boolean equals(PublicKey other) {
 return Arrays.equals(modulus, other.modulus)
   && Arrays.equals(pubexponent, other.pubexponent);
}
public byte getCode() {
 return Code.PUBLICKEY;
}
≪IF targetServer()||targetTerminal()≫
public void setKey(byte[] modulus) {
 this.modulus=modulus.clone();
}≪ENDIF≫
```

```
public void copy(PublicKey p) {
≪IF targetCard()≫
Arrays.copy(p.modulus, modulus);
 Arrays.copy(p.pubexponent, pubexponent);
 updateKey();
≪ELSE≫
if (p.getModulus() != null) {
  setKey(p.getModulus());
 }
≪ENDIF≫
}
public boolean isPublicKey() {
 return true;
}
≪IF targetCard()≫
public void updateKey() {
 apikey
   .setExponent(pubexponent,(short)0,EXPONENTLENGTH);
 apikey.setModulus(modulus,(short)0,MODULUSLENGTH);
}
public javacard.security.Key toAPIKey() {
 return apikey;
}
≪ELSE≫
public java.security.PublicKey toAPIKey()
  throws Exception{
 BigInteger mod = new BigInteger(1, this.modulus);
 BigInteger pub = new BigInteger(1, this.pubexponent);
 try{
  KeyFactory rsakf = KeyFactory
    .getInstance("RSA", cryptProvider);
  java.security.PublicKey apikey = (java.security
    .PublicKey) rsakf
    .generatePublic(new java.security.spec
      .RSAPublicKeySpec(mod,
      pub));
  return apikey;
  }
  catch(Exception e){
   return null;
  }
}
public byte[] getModulus() {
 return modulus;
}
public byte[] getPubexponent() {
 return pubexponent;
}
≪ENDIF≫
}
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generatePrivateKeyClass FOR Model≫
```

```
≪FILE this.getModel().getDirectoryPrefix()
  +privkeyName()+".java"−≫
package ≪this.packagename()−≫;
≪IF targetCard()≫
import javacard.security.RSAPrivateKey;
import javacard.security.KeyBuilder;
≪ELSE≫
import java.math.BigInteger;
import java.security.spec.InvalidKeySpecException;
import java.security.*;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
@XmlAccessorType(XmlAccessType.FIELD)≪ENDIF≫
public class PrivateKey extends Key ≪this
  .ppImplements()≫ {
public final static short MODULUSLENGTH = 128;
public final static short EXPONENTLENGTH = 128;
≪IF targetServer()||targetTerminal()≫
 public final static String cryptProvider = "BC";
≪ELSE≫
private RSAPrivateKey apikey;
≪ENDIF≫
≪IF targetCard()≫
public≪ELSE≫private≪ENDIF≫ byte[] modulus;
≪IF targetCard()≫
public≪ELSE≫private≪ENDIF≫ byte[] privexponent;
private static final byte[] defaultModulus = ≪defaultPubModulus
  ()≫;
private static final byte[] defaultExponent = ≪defaultPrivExponent
  ()≫;
public PrivateKey(){
≪IF targetCard()≫
modulus = new byte[MODULUSLENGTH];
 privexponent = new byte[EXPONENTLENGTH];
 apikey = (RSAPrivateKey)KeyBuilder
   .buildKey(KeyBuilder.TYPE_RSA_PRIVATE,
        KeyBuilder.LENGTH_RSA_1024,
        false);
 Arrays.copy(defaultModulus, this.modulus);
 Arrays.copy(defaultExponent, privexponent);
 updateKey();
 ≪ELSE≫
 Security.addProvider(new org.bouncycastle.jce
   .provider.BouncyCastleProvider());
 setKey(defaultModulus, defaultExponent);
 ≪ENDIF≫
}
public PrivateKey(byte[] modulus, byte[] privexp) {
 this();
 ≪IF targetServer()||targetTerminal()≫
 setKey(modulus, privexp);≪ELSE≫
 Arrays.copy(modulus, this.modulus);
 Arrays.copy(privexp, privexponent);
 updateKey();
```

```
 ≪ENDIF≫
}
public boolean equals(PrivateKey other) {
 return Arrays.equals(modulus, other.modulus)
   && Arrays.equals(privexponent, other.privexponent);
}
public byte getCode() {
 return Code.PRIVATEKEY;
}
≪IF targetServer()||targetTerminal()≫
public void setKey(byte[] modulus, byte[] privexp) {
 this.modulus=modulus.clone();
 this.privexponent=privexp.clone();
}
public java.security.PrivateKey toAPIKey()
  throws Exception{
 BigInteger mod = new BigInteger(1, this.modulus);
 BigInteger priv = new BigInteger(1, this
   .privexponent);
 try{
  KeyFactory rsakf = KeyFactory
    .getInstance("RSA", cryptProvider);
  java.security.PrivateKey apikey = (java.security
    .PrivateKey) rsakf
    .generatePrivate(new java.security.spec
      .RSAPrivateKeySpec(mod,
      priv));
  return apikey;
  }
  catch(Exception e){
   return null;
  }
}
public byte[] getModulus() {
 return modulus;
}
public byte[] getPrivexponent() {
 return privexponent;
}
≪ELSE≫
public void updateKey() {
 apikey
   .setExponent(privexponent,(short)0,EXPONENTLENGTH);
 apikey.setModulus(modulus,(short)0,MODULUSLENGTH);
}
public javacard.security.Key toAPIKey() {
 return apikey;
}
≪ENDIF≫
public void copy(PrivateKey p) {
 ≪IF targetCard()≫
 Arrays.copy(p.modulus, modulus);
 Arrays.copy(p.privexponent, privexponent);
 updateKey();
```

```
 ≪ELSE≫
 if (p.getModulus() != null && p
   .getPrivexponent() != null) {
  setKey(p.getModulus(), p.getPrivexponent());
 }≪ENDIF≫
}
public boolean isPrivateKey() {
 return true;
}
}
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateHashedDataClass FOR Model≫
≪FILE this.getModel().getDirectoryPrefix()
  +hashedDataName()+".java"−≫
 package ≪this.packagename()−≫;
≪IF targetCard()≫
import javacard.framework.*;
 import javacard.security.MessageDigest;
 ≪ELSE≫
 import java.security.MessageDigest;
 import java.security.NoSuchAlgorithmException;
 import java.security.NoSuchProviderException;
 import swt.util.ByteArray;
 import java.security.Security;
    import javax.xml.bind.annotation.XmlAccessType;
 import javax.xml.bind.annotation.XmlAccessorType;
 @XmlAccessorType(XmlAccessType.FIELD)≪ENDIF≫
 public class HashedData ≪this
   .ppImplements()≫ {
 public final static short HASHLENGTH = 20;
 public byte[] hashed;
 ≪IF targetServer()||targetTerminal()≫
 public final static String cryptProvider = "BC";
 private byte[] tohash;
 ≪ELSE≫
 private static MessageDigest md;
 private static byte[] tohash = new byte[Store
   .COMPUTEDHASHDATALENGTH];
 ≪ENDIF≫
 public HashedData() {
 ≪IF targetServer()||targetTerminal()≫
     Security.addProvider(new org.bouncycastle.jce
       .provider.BouncyCastleProvider());
  tohash = new byte[≪lengthConstantsName()≫
    .COMPUTEDHASHDATALENGTH];
 ≪ENDIF≫
  hashed = new byte[HASHLENGTH];
 }
≪IF targetServer()||targetTerminal()≫
public byte[] getHashed() {
  return hashed;
 }
```

```
 public void setHashed(byte[] hashed) {
 ByteArray.copy(hashed, 0, this.hashed, 0,
   (short) this.hashed.length);
 }
≪ENDIF≫
 public void copy(HashedData from) {
 ≪IF targetCard()≫
   Arrays.copy(from.hashed, this.hashed);
 ≪ELSE≫
 this.hashed=from.getHashed().clone();
 ≪ENDIF≫
}
public byte getCode() {
 return Code.HASHEDDATA;
}
public boolean equals(HashedData other) {
 return Arrays.equals(hashed, other.hashed);
}
public static HashedData hash
  (HashData h) ≪IF targetServer()||targetTerminal
    ()≫
      throws ≪exceptionName
        ()≫ ≪ENDIF≫{
 ≪IF targetCard()≫
 if (md == null) md = MessageDigest
   .getInstance(MessageDigest.ALG_SHA, false);
  return Store.newHashedData().insthash(h);
 ≪ELSE≫
 return (new HashedData()).insthash(h);
 ≪ENDIF≫}
public HashedData insthash
  (HashData h) ≪IF targetServer()||targetTerminal
    ()≫
      throws ≪exceptionName
        ()≫ ≪ENDIF≫{
 Coding c = Coding.getInstance();
 c.encode(h, tohash);
 ≪IF targetCard()≫
 md.doFinal(tohash, (short)0, c
   .getEncodingLength(), hashed, (short)0);
 ≪ELSE≫
 try{
 MessageDigest md = MessageDigest
   .getInstance("SHA1", cryptProvider);
 hashed = md
   .digest(ByteArray.subarray(tohash, 0, c
     .getEncodingLength()));
 }
 catch(Exception e){
  return null;
 }
 ≪ENDIF≫
 return this;
}
```

```
}
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generatePlainDataClass FOR Model≫
≪FILE getDirectoryPrefix()+plainDataName()
  +".java"−≫
package ≪this.packagename()−≫;
public interface ≪plainDataName
  ()≫ extends ≪codeableName()≫ {
public byte getCode();
}
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateEncDataClass FOR Model≫
≪FILE this.getModel().getDirectoryPrefix()
  +encDataName()+".java"−≫
package ≪this.packagename()−≫;
≪IF targetCard()≫
import javacard.framework.Util;
import javacardx.crypto.Cipher;
≪ELSE≫
import javax.crypto.Cipher;
import javax.xml.bind.annotation.*;
@XmlAccessorType(XmlAccessType.FIELD)≪ENDIF≫
public abstract class ≪encDataName
  ()−≫ ≪this.ppImplements()≫ {
≪IF targetCard
  ()≫ public ≪ELSE≫protected≪ENDIF≫ byte[] encrypted;
≪IF targetServer()||targetTerminal()≫
protected int plainlength;
protected int enclength;
≪ELSE≫
public short plainlength;
public short enclength;
≪ENDIF≫
≪IF targetCard
  ()≫ public ≪ELSE≫protected≪ENDIF≫ byte plainDataType;
protected static byte[] tmparray;
≪IF !targetCard()≫
@XmlTransient
≪ENDIF≫
protected Cipher c;
public boolean is≪encDataSymmName()≫(){
 return false;
}
public boolean is≪encDataAsymmName()≫(){
 return false;
}
≪IF targetServer()||targetTerminal()≫
public byte[] getEncrypted() {
 return encrypted;
}
```

```
public void setEncrypted(byte[] src) {
 encrypted=src.clone();
}
«ENDIF»
public void copy(«encDataName()» o) {
 enclength = o.enclength;
 plainlength = o.plainlength;
 plainDataType = o.plainDataType;
 «IF targetCard()»
 Arrays.copy(o.encrypted, encrypted);
 «ELSE»o.setEncrypted(encrypted);
 «ENDIF»}
public boolean equals(«encDataName()» o) {
 if (enclength != o.enclength || plainlength != o
   .plainlength
   || plainDataType != o.plainDataType)
  return false;
 «IF targetCard()»
 return Arrays.equals(encrypted, o.encrypted);
 «ELSE»
 return Arrays.equals(encrypted, o
   .getEncrypted());«ENDIF»
}
}
«ENDFILE»
«ENDDEFINE»

«DEFINE generateCopyNewObject FOR Class»
public «this.name
  .toString()» «copyName()»() {
 «this.name.toString()» object = new «this
   .name.toString()»();
 object.«copyName()»(this);
 return object;
}
«ENDDEFINE»

«DEFINE generateEncDataSymmClass FOR Model»
«FILE this.getModel().getDirectoryPrefix()
  +encDataSymmName()+".java"-»
package «this.packagename()-»;
«IF targetCard()»
import javacard.framework.*;
import javacardx.crypto.Cipher;
«ELSE»
import java.security.*;
import java.security.spec.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlTransient;
@XmlAccessorType(XmlAccessType.FIELD)«ENDIF»
public class «encDataSymmName
```

```
  ()≫ extends ≪encDataName()≫{
≪IF targetServer()||targetTerminal()≫
private final static String cryptProvider = "BC";
private final static byte[] iv = {0,0,0,0,0,0,0,0};
@XmlTransient
private final static AlgorithmParameterSpec paramSpec = new IvParameterSpec
  (iv);≪ENDIF≫
public ≪encDataSymmName()≫(){
 encrypted = new byte[≪IF targetCard
   ()≫Store≪ELSE≫≪lengthConstantsName
     ()≫≪ENDIF≫.MAXENCRYPTLENGTHSYMM];
 tmparray = new byte[≪IF targetCard
   ()≫Store≪ELSE≫≪lengthConstantsName
     ()≫≪ENDIF≫.MAXENCRYPTLENGTHSYMM];
 ≪IF targetCard()≫
 c = Cipher.getInstance(
   Cipher.ALG_DES_CBC_NOPAD, false);
 ≪ELSE≫
   Security.addProvider(new org.bouncycastle.jce
     .provider.BouncyCastleProvider());
 ≪ENDIF≫
}
public byte getCode() {
 return ≪codeName()≫.≪encDataSymmName()
   .toString().toUpperCase()≫;
}
public boolean is≪encDataSymmName()≫(){
 return true;
}
public static ≪encDataSymmName()≫ encrypt
  (SymmKey key, PlainData d) ≪IF targetServer
    ()||targetTerminal()≫
      throws ≪exceptionName()≫≪ENDIF≫{
 ≪IF targetServer()||targetTerminal()≫
 return (new ≪encDataSymmName()≫())
   .instEncrypt(key, d);
 ≪ELSE≫
 return Store.new≪encDataSymmName()≫()
   .instEncrypt(key, d);
 ≪ENDIF≫
}
public ≪encDataSymmName()≫ instEncrypt
  (SymmKey key, PlainData d) ≪IF targetServer
    ()||targetTerminal()≫
      throws ≪exceptionName()≫≪ENDIF≫ {
 plainDataType = d.getCode();
 ≪IF targetCard()≫
 Util
   .arrayFillNonAtomic(tmparray, (short)0,
     (short)tmparray.length, (byte)0x0);
 ≪ELSE≫
 java.util.Arrays.fill(tmparray, (byte)0);
 ≪ENDIF≫
 ≪codingName()≫.getInstance()
```

```
    .encode(d, tmparray);
  plainlength = ≪codingName()≫.getInstance()
    .getEncodingLength();
  enclength = plainlength % 8 == 0
    ? plainlength
    : (short) (8 + plainlength - plainlength % 8);
    ≪IF targetCard()≫
    c.init(key.toAPIKey(), Cipher.MODE_ENCRYPT);
    c
      .doFinal(tmparray,
        (short) 0, enclength, encrypted, (short) 0);
    return this;
    ≪ELSE≫
    try{
      c = Cipher
        .getInstance
          ("DESede/CBC/NoPadding", cryptProvider);
      c.init(Cipher.ENCRYPT_MODE, key
        .toAPIKey(), paramSpec);
      c.doFinal(tmparray, 0, enclength, encrypted, 0);
      return this;
     }
     catch(Exception e)
     {
      return null;
     }
    ≪ENDIF≫
}
private boolean check() {
 if (!
    (0 < plainlength && plainlength <= enclength && enclength <= encrypted
      .length))
   return false;
   return enclength % 8 == 0;
}
public static PlainData decrypt
  (SymmKey key, ≪encDataSymmName
    ()≫ e)≪IF targetServer()||targetTerminal
      ()≫
        throws ≪exceptionName
          ()≫≪ENDIF≫{
 return e.decrypt(key);
}
public PlainData decrypt
  (SymmKey key) ≪IF targetServer()||targetTerminal
    ()≫
      throws ≪exceptionName()≫≪ENDIF≫ {
 if (!check())
  ≪IF targetCard()≫
  ≪abortClassName()≫();≪ELSE≫
  ≪abortName()≫();≪ENDIF≫
  ≪IF targetServer()||targetTerminal()≫
  try{
  c = Cipher
```

```
    .getInstance
      ("DESede/CBC/NoPadding", cryptProvider);
  c.init(Cipher.DECRYPT_MODE, key
    .toAPIKey(), paramSpec);
  c.doFinal(encrypted, 0, enclength, tmparray, 0);
  }
  catch(Exception e){
   stop();
   return null;
  }
 ≪ELSE≫
 c.init(key.toAPIKey(), Cipher.MODE_DECRYPT);
 c
   .doFinal(encrypted,
     (short) 0, enclength, tmparray, (short) 0);
 ≪ENDIF≫
 return ≪codingName()≫.getInstance()
   .decodePlainData(tmparray, plainDataType);
}
≪IF targetServer()||targetTerminal()≫
private void ≪abortName()≫()
  throws ≪exceptionName()≫{
   throw new ≪exceptionName()≫
     ("≪encDataSymmName()≫");
  }
≪EXPAND generateCopyNewObject
  FOR this.getEncDataSymm()≫
≪ENDIF≫
}
≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generateEncDataAsymmClass FOR Model≫
≪FILE this.getModel().getDirectoryPrefix()
  +encDataAsymmName()+".java"−≫
package ≪this.packagename()−≫;
≪IF targetCard()≫
import javacard.framework.*;
import javacard.security.RandomData;
import javacardx.crypto.Cipher;
≪ELSE≫
import java.security.*;
import javax.crypto.*;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
@XmlAccessorType(XmlAccessType.FIELD)≪ENDIF≫
public class ≪encDataAsymmName
  ()≫ extends ≪encDataName()≫{
private final static short RSABLOCKLENGTH = 128;
≪IF targetCard()≫
 private static RandomData rd;
≪ELSE≫
 private final static String cryptProvider = "BC";
≪ENDIF≫
```

```
public ≪encDataAsymmName()≫(){
 encrypted = new byte[≪IF targetServer
   ()||targetTerminal()≫≪lengthConstantsName
     ()≫≪ELSE≫Store≪ENDIF≫
       .MAXENCRYPTLENGTHASYMM];
 tmparray = new byte[≪IF targetServer
   ()||targetTerminal()≫≪lengthConstantsName
     ()≫≪ELSE≫Store≪ENDIF≫
       .MAXENCRYPTLENGTHASYMM];
 ≪IF targetServer()||targetTerminal()≫
 Security.addProvider(new org.bouncycastle.jce
   .provider.BouncyCastleProvider());
 ≪ELSE≫
 c = Cipher.getInstance(Cipher.ALG_RSA_NOPAD,
   false);
 if (rd == null)
  rd = RandomData.getInstance(RandomData
    .ALG_SECURE_RANDOM);
 ≪ENDIF≫
 }
public byte getCode() {
 return ≪codeName()≫.≪encDataAsymmName()
   .toString().toUpperCase()≫;
}
public boolean is≪encDataAsymmName()≫(){
 return true;
}
 public static ≪encDataAsymmName()≫ encrypt
   (PublicKey key, PlainData d)≪IF targetTerminal
     ()||targetServer()≫
       throws ≪exceptionName
         ()≫≪ENDIF≫ {
 return ≪IF targetCard()≫Store
   .new≪encDataAsymmName()≫()
     .instEncrypt(key, d);≪ELSE≫
       (new ≪encDataAsymmName()≫())
         .instEncrypt(key, d);≪ENDIF≫
}
public ≪encDataAsymmName()≫ instEncrypt
  (PublicKey key, PlainData d)≪IF targetTerminal
    ()||targetServer()≫
      throws ≪exceptionName()≫≪ENDIF≫ {
 plainDataType = d.getCode();
 ≪codingName()≫.getInstance()
   .encode(d, tmparray);
 plainlength = ≪codingName()≫.getInstance()
   .getEncodingLength();
 enclength = RSABLOCKLENGTH;
 ≪IF targetCard()≫
 padBuffer(tmparray, (short)0, plainlength, enclength);
 c.init(key.toAPIKey(), Cipher.MODE_ENCRYPT);
 c
   .doFinal(tmparray,
     (short) 0, plainlength, encrypted, (short) 0);
```

```
 ≪ELSE≫
 try{
  c = Cipher.getInstance("RSA", cryptProvider);
  c.init(Cipher.ENCRYPT_MODE, key.toAPIKey());
  c.doFinal(tmparray, 0, enclength, encrypted, 0);
  }
  catch(Exception e){
   return null;
  }
 ≪ENDIF≫
 return this;
}
 ≪IF targetCard()≫
private void padBuffer
  (byte[] buffer, short offset, short len
    , short paddedlen){
 if(paddedlen != len)
  rd.generateData(buffer, (short)(offset
    +len), (short)(paddedlen-len));
}
 ≪ENDIF≫
private boolean check() {
 if (!
   (0 < plainlength && plainlength <= enclength && enclength <= encrypted
     .length))
  return false;
  return enclength == RSABLOCKLENGTH;
}
public static PlainData decrypt
  (PrivateKey key, ≪encDataAsymmName
    ()≫ e) ≪IF targetServer()||targetTerminal
      ()≫
        throws ≪exceptionName
          ()≫ ≪ENDIF≫{
 return e.decrypt(key);
}
public PlainData decrypt
  (PrivateKey key)≪IF targetServer
    ()||targetTerminal()≫
      throws ≪exceptionName
        ()≫ ≪ENDIF≫ {
 if (!check())
  ≪IF targetCard()≫
  ≪abortClassName()≫();
  ≪ELSE≫≪abortName()≫
    (); ≪ENDIF≫
  ≪IF targetCard()≫
  c.init(key.toAPIKey(), Cipher.MODE_DECRYPT);
  c
    .doFinal(encrypted,
      (short) 0, enclength, tmparray, (short) 0);
  ≪ELSE≫
  try{
  c = Cipher.getInstance("RSA", cryptProvider);
```

```
  c.init(Cipher.DECRYPT_MODE, key.toAPIKey());
  c.doFinal(encrypted, 0, enclength, tmparray, 0);
  }
  catch(Exception e){
   stop();
   return null;
  }
  «ENDIF»
 return «codingName()».getInstance()
   .decodePlainData(tmparray, plainDataType);
}
«IF targetServer()||targetTerminal()»
private static void «abortName()»()
  throws «exceptionName()»{
 throw new «exceptionName()»
   ("«encDataAsymmName()»");
}
 «EXPAND generateCopyNewObject
   FOR this.getEncDataAsymm()»
«ENDIF»
}
«ENDFILE»
«ENDDEFINE»

«DEFINE generateMACDataClass FOR Model»
«FILE this.getModel().getDirectoryPrefix()
  +macDataName()+".java"»
 package «this.packagename()-»;
 «IF targetCard()»
 import javacard.framework.*;
 «ELSE»
 import javax.xml.bind.annotation.XmlAccessType;
 import javax.xml.bind.annotation.XmlAccessorType;
 @XmlAccessorType(XmlAccessType.FIELD)
 «ENDIF»
 public class «macDataName()» «this
   .ppImplements()»{
  «encDataSymmName()» encData;
  «encDataSymmName()» mac;
  public «macDataName()»(){
   encData= new «encDataSymmName()»();
   mac= new «encDataSymmName()»();
  }
  public static «macDataName()» computeMAC
    («symmkeyName()» encKey, «symmkeyName
      ()» macKey, «plainDataName
        ()» pd) «IF targetServer
          ()||targetTerminal()»
            throws «exceptionName
              ()» «ENDIF»{
   «macDataName()» md= «IF targetCard
     ()» «storeName()»
       .new«macDataName()»
         ()«ELSE»new «macDataName()»
```

```
            ()≪ENDIF≫;
    md.encData= ≪encDataSymmName()≫
      .encrypt(encKey, pd);
    ≪hashedDataName
      ()≫ hash = ≪hashedDataName()≫.hash(md
        .encData);
    md.mac= ≪encDataSymmName()≫
      .encrypt(macKey, hash) ;
    return md;
   }
   public static ≪plainDataName()≫ decryptMAC
     (≪symmkeyName()≫ encKey, ≪symmkeyName
       ()≫ macKey, ≪macDataName
         ()≫ md)≪IF targetServer
           ()||targetTerminal()≫
             throws ≪exceptionName
               ()≫≪ENDIF≫{
    ≪hashedDataName
      ()≫ hash = ≪hashedDataName()≫.hash(md
        .encData);
    ≪encDataSymmName
      ()≫ mac = ≪encDataSymmName()≫
        .encrypt(macKey, hash);
    if( !mac.equals( md.mac ) )
     ≪IF targetCard()≫
     ≪abortClassName()≫();
     ≪ELSE≫
     ≪abortName()≫();
     ≪ENDIF≫
    return ≪encDataSymmName()≫
      .decrypt(encKey, md.encData);
   }
  ≪IF targetServer()||targetTerminal()≫
  private static void ≪abortName()≫()
    throws ≪exceptionName()≫{
    throw new ≪exceptionName()≫
      ("≪macDataName()≫");
   }≪ENDIF≫
public byte getCode() {
    return ≪codeName()≫.≪macDataName()
      .toString().toUpperCase()≫;
   }
   public boolean is≪macDataName()≫(){
    return true;
   }
   public void copy(≪macDataName()≫ o) {
    encData.copy(o.encData);
    mac.copy(o.mac);
   }
   public boolean equals(≪macDataName()≫ o) {
    return o.encData.equals(encData) && o.mac
      .equals(mac);
   }
   ≪IF targetServer()||targetTerminal()≫
```

```
  public «encDataSymmName()» getEncData(){
   return encData;
  }
  public «encDataSymmName()» getMac(){
   return mac;
  }
  public void setEncData(«encDataSymmName
    ()» encData){
   this.encData=encData;
  }
  public void setMac(«encDataSymmName()» mac){
   this.mac=mac;
  }
  «EXPAND generateCopyNewObject
    FOR this.getMACData()»
 «ENDIF»
 }
«ENDFILE»
«ENDDEFINE»

«DEFINE generateCodeableClass FOR Class»
«LET this.getModel() AS mo»
«FILE mo.getDirectoryPrefix()+codeableName()
  +".java"»
 package «this.getModel().packagename()-»;
 public interface «codeableName()» {
  public byte getCode() ;
 }
«ENDFILE»
«ENDLET»
«ENDDEFINE»

«DEFINE generateCodeClass FOR Class»
«LET this.getModel() AS mo»
«FILE mo.getDirectoryPrefix()+codeName()
  +".java"»
 package «this.getModel().packagename()-»;
 public class «codeName()» {
 «FOREACH mo.getClass(codeName()).ownedAttribute
   AS prop-»
  «prop.getAttributeImplCode()»
 «ENDFOREACH»
}
«ENDFILE»
«ENDLET»
«ENDDEFINE»

«DEFINE generateMath FOR Class»
«IF targetCard()»
«FILE this.getModel().getDirectoryPrefix()
  + mathName()+".java"-»
«EXPAND generatedPackage FOR this»
import javacard.framework.ISOException;
import javacard.framework.ISO7816;
```

```
«EXPAND generateClassContructors FOR this-»{
   «FOREACH this.ownedOperation AS op-»
«LET op.ownedParameter.reject(e|e.name==null||e
  .name==this.name) AS param»
 «EXPAND generateClassConstructor(op)
   FOR this-»{
  «IF op.name=="plus"»
     short res = (short)(x + y);
  if (x < 0 && y < 0 && res >= 0) ISOException
    .throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
  if (x > 0 && y > 0 && res < 0) ISOException
    .throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
  return res;
  «ELSEIF op.name=="overflowsPlus"»
     short res = (short) (x + y);
    if (x < 0 && y < 0 && res >= 0)
     return true;
    if (x > 0 && y > 0 && res < 0)
     return true;
    return false;
  «ELSEIF op.name=="minus"»
     short res = (short)(x - y);
       if (0 <= x && y < 0 && res < 0)
    ISOException.throwIt(ISO7816
      .SW_CONDITIONS_NOT_SATISFIED);
       if (x < 0 && 0 < y && res > 0)
    ISOException.throwIt(ISO7816
      .SW_CONDITIONS_NOT_SATISFIED);
       return res;
  «ELSEIF op.name=="div"»
     if (y == 0) ISOException.throwIt(ISO7816
       .SW_CONDITIONS_NOT_SATISFIED);
    if (y == -1 && x == -32768) ISOException
      .throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
    return (short)(x / y);
 «ELSEIF op.name=="rem"»
    if (y == 0) ISOException.throwIt(ISO7816
      .SW_CONDITIONS_NOT_SATISFIED);
    return (short)(x % y);
  «ELSEIF op.name=="mult"»
     if (x == 0 || y == 0) return 0;
    if (x == 1) return y;
    if (y == 1) return x;
    if (x == -32768 || y == -32768) ISOException
      .throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
    byte sgn = ( x < 0 ? (y < 0 ? 1 : (byte)-1) :
      (y < 0 ? (byte)-1 : 1));
    if (x < 0) x = (short)- x;
    if (y < 0) y = (short)- y;
    short xh = (short)(x >> 8);
    short yh = (short)(y >> 8);
    if (xh != 0 && yh != 0) ISOException
      .throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
    short xl = (short)(x & 0xFF);
```

```
    short yl = (short)(y & 0xFF);
    short r1 = (short)(xl * yl);
    short r2 = (short)(xl * yh);
    short r3 = (short)(xh * yl);
    short r = (short)(r2 + r3);
    short res = (short)(r * 256 + r1);
    if (sgn < 0 && r == 128 && r1 == 0) return -32768;
    if (r > 127 || res < 0) ISOException
      .throwIt(ISO7816.SW_CONDITIONS_NOT_SATISFIED);
    return (short)(sgn * res);
  «ENDIF»
}
«ENDLET»
«ENDFOREACH»
}
«ENDFILE»
«ENDIF»
«ENDDEFINE»

«DEFINE generateSimpleComm (Class c) FOR Class»
«FILE this.getModel().getDirectoryPrefix()
  + simpleCommName()+".java"-»
«EXPAND generatedPackage FOR this»
  import javacard.framework.APDU;
  import javacard.framework.Applet;
  import javacard.framework.ISO7816;
  import javacard.framework.ISOException;
  import javacard.framework.Util;
«EXPAND generateClassContructors
  FOR this-» extends Applet{
   «FOREACH this.instanceAttributes() AS p»
    «p.getAttributeImplCode()»
   «ENDFOREACH»
   private «this.getModel()
     .getNameMessageSuperclass()» inmsg;
   private static «this.getModel()
     .getNameMessageSuperclass()» outmsg;
   protected byte[] encodebuffer;
   protected byte[] decodebuffer;
   private short len = 0;
   private static short encbuf_sentlen = 0;
   private static short decbuf_receivedlen = 0;
   public byte[] prevMsg;
   public short prevMsgLen;
   public static «gateName()» dummyGate;
   public static «gateName()» currentGate;
 «FOREACH this.ownedOperation AS op-»
  «LET op.ownedParameter.reject(e|e.name==null||e
    .name==this.name) AS param»
   «IF op.name.startsWith("reset")»
     «EXPAND generateClassConstructor(op)
       FOR this»{
       currentGate = dummyGate;
     }
```

```
≪ENDIF≫
≪IF op.name.startsWith("set")≫
 ≪EXPAND generateClassConstructor(op)
   FOR this≫{
  currentGate = gate;
 }
≪ENDIF≫
≪IF op.name.toString()=="process"≫
   ≪IF param.name.first()=="apdu"≫
    ≪EXPAND generateClassConstructor(op)
      FOR this≫{
     if (selectingApplet())
    return;
   short expected = apdu.setIncomingAndReceive();
   if (expected == 0)
    ISOException.throwIt(ISO7816
       .SW_CONDITIONS_NOT_SATISFIED);
   byte ins = apdu.getBuffer()[ISO7816.OFFSET_INS];
   if (ins == RESUME_INSTRUCTION){
    currentGate
       .sendAPDU(this, apdu, encodebuffer, (short)
         (len-encbuf_sentlen), false);
    return;
   }
   if((short)(decbuf_receivedlen
     + expected) > DECBUF_MAXLEN){
    decbuf_receivedlen = 0;
    ISOException.throwIt(ISO7816
       .SW_CONDITIONS_NOT_SATISFIED);
    return;
   }
   Util.arrayCopy(apdu.getBuffer(), ISO7816
     .OFFSET_CDATA, decodebuffer
       , decbuf_receivedlen, expected);
   decbuf_receivedlen += expected;
   if(apdu.getBuffer()[ISO7816
     .OFFSET_P1] == 1) return;
   if (ins == INIT_INSTRUCTION){
    appletInitialization(decodebuffer, (short)0);
    decbuf_receivedlen = 0;
    return;
   }
   Store.reset();
   outmsg = null;
   prevMsg = decodebuffer;
   prevMsgLen = decbuf_receivedlen;
   inmsg = coding
     .decodeMessage(decodebuffer, (short)0,
     decbuf_receivedlen);
   decbuf_receivedlen = 0;
   process(inmsg);
   if (outmsg != null) {
    len = coding
       .encode((Codeable) outmsg, encodebuffer);
```

```
    encbuf_sentlen = 0;
    currentGate
      .sendAPDU
        (this, apdu, encodebuffer, len, true);
   }
   else
    currentGate.finishProtocol(this);
    }
     ≪ELSE≫ ≪EXPAND generateClassConstructor
       (op) FOR this≫;
     ≪ENDIF≫
≪ENDIF≫
≪IF op.name.toString().contains("sendAPDU")≫
 ≪EXPAND generateClassConstructor(op)
   FOR this≫{
    if(!first && encbuf_sentlen >= len)
     return;
    byte[] buffer = apdu.getBuffer();
    short sendLen = 0;
    if(rem_len > OUTMSG_MAXLEN){
     buffer[0] = 1;
     Util
       .arrayCopy(data, encbuf_sentlen, buffer,
         (short) 1, OUTMSG_MAXLEN);
     encbuf_sentlen += OUTMSG_MAXLEN;
     sendLen = APDU_MAXLEN;
    }
    else{
     buffer[0] = 0;
     Util
       .arrayCopy(data, encbuf_sentlen, buffer,
         (short) 1, rem_len);
     encbuf_sentlen = len;
     sendLen = (short)(rem_len + 1);
    }
    apdu.setOutgoingAndSend((short) 0, sendLen);
    }
 ≪ENDIF≫
 ≪IF op.name
   .toString
     ()=="appletInitialization"≫≪EXPAND generateClassConstructor
       (op) FOR this≫;
 ≪ENDIF≫
 ≪IF op.name
   .toString
     ()=="sendMsg"≫ ≪EXPAND generateClassConstructor
       (op) FOR this≫{
   currentGate.callConstraints();
   outmsg = msg;
   }
 ≪ENDIF≫
 ≪IF op.name=="SimpleComm"≫
   ≪IF param.type.name.first()!="byte"≫
   public ≪simpleCommName()-≫() {
```

```
        coding = ≪codingName()≫.getInstance();
        encodebuffer = new byte[ENCBUF_MAXLEN];
        decodebuffer = new byte[DECBUF_MAXLEN];
        Store.initAll();
        register();
       }
       ≪ENDIF≫
      ≪ENDIF≫
      ≪IF op.name.toString()=="install"≫
       ≪EXPAND generateClassConstructor(op)
        FOR this≫{
        new ≪c.name≫();
        dummyGate = new Dummy≪gateName()≫();
        currentGate = dummyGate;
       }
      ≪ENDIF≫
      ≪IF op.name.toString()=="stop"≫
       ≪EXPAND generateClassConstructor(op)
        FOR this≫{
        currentGate.callConstraints();
        ≪FOREACH this.getStructuredInitNodes()
          .select(e|e
            .hasStereotype("SecureMDD::Exception"))
              AS exceptionMethod≫
        ≪IF exceptionMethod.inPartition.first()
          .getPartitionClass()==this.name.toString() ||
            ((! this.superClass
              .isEmpty)?(exceptionMethod.inPartition
                .first().getPartitionClass()==this
                  .superClass.first().name
                    .toString()):
                      (false)) || exceptionMethod.owner
                        .metaType==StructuredActivityNode-≫
        ≪exceptionMethod.name≫();
        ≪ENDIF≫
        ≪ENDFOREACH-≫
        ISOException.throwIt(ISO7816
          .SW_CONDITIONS_NOT_SATISFIED);
       }
      ≪ENDIF≫
≪ENDLET≫
≪ENDFOREACH≫
}
 ≪ENDFILE≫
≪ENDDEFINE≫

≪DEFINE generatedPackage FOR Class≫
   package ≪this.getModel().packagename()-≫;
≪ENDDEFINE≫

≪DEFINE generateClassConstructor(Operation op)
  FOR Class≫
≪IF op.ownedParameter!=null≫
 ≪op.visibility.toString()≫ ≪IF op
```

```
    .isStatic≫ static ≪ELSEIF op.isAbstract()≫abstract≪ENDIF≫
    ≪IF op.type.name==null≫void ≪ELSE≫≪op.type.name≫ ≪ENDIF≫
    ≪op.name≫ (≪FOREACH op
                .ownedParameter.reject(e|e
                  .name==null)
                    AS param SEPARATOR ”
                      , ” ≫ ≪IF param
                        .type!=null≫ ≪IF param
                          .translateType()==null≫
                            ≪param.type.name≫≪ELSE≫≪param
                                .translateType()≫≪ENDIF≫ ≪ENDIF≫
                                ≪param.name≫≪ENDFOREACH≫)
≪ENDIF≫
≪ENDDEFINE≫

≪DEFINE generateClassContructors FOR Class≫
≪this.visibility.toString()≫ ≪IF this
  .isAbstract≫ abstract ≪ENDIF≫class ≪this
    .name.toString().toFirstUpper()≫
≪ENDDEFINE≫

≪DEFINE blub FOR Model≫
≪FILE this.getModel().getDirectoryPrefix()
  + ”NameDerDatei”+”.java”−≫
package ≪this.getModel().packagename()−≫;
≪ENDFILE≫
≪ENDDEFINE≫
```

3.3.6 Operations that execute several tests on the source model

```
import uml;
extension psm_templates_shared::names;
extension psm_templates_shared::getter;
extension psm_templates_shared::translations;
extension psm_templates_shared::parse;
Boolean isDebugMode() : true;

Boolean targetCard() :
 ((String)GLOBALVAR isJavaCard).matches(”true”)
   ? true : false;

Boolean targetTerminal() :
 ((String)GLOBALVAR isTerminal).matches(”true”)
   ? true : false;

Boolean targetServer() :
 ((String)GLOBALVAR isServer).matches(”true”)
   ? true : false;

Boolean targetTestcase() :
 ((String)GLOBALVAR isTestcase).matches(”true”)
   ? true : false;
```

```
Boolean targetAndroid() :
 false;

Boolean isUser(Property prop) :
 prop.type == prop.getModel().getUserClass();

Boolean isUser(Lifeline ll) :
 ll.represents.type == ll.getModel().getUserClass();

Boolean isUser(Type ty) :
 ty == ty.getModel().getUserClass();

Boolean isUsedCodeableClass(Class cla) :
 if (cla == null) then false else
 getClass(cla.getModel(),storeName
    ()).ownedAttribute.exists(o|o.name.split
      ("MaxCountFor").get(0)==cla.name &&
        o.name.split("MaxCountFor").get(1)
          ==cla.getModel().getGeneratingCard()
       );

Boolean isListClass(Class c) :
 (c.getModel().getStarAssociations().exists
    (e|c.name.toString().matches
      ("ListOf"+e.name.toString())));

Boolean isCustom(Property e, Class cla) :
 getCustomCardClassAttributes(cla).contains
    (e.name.toString()) ;

Boolean implementsInterface(Class c, String i) :
 c.getAllImplementedInterfaces().exists(e|e.name == i);
cached Boolean requiresEncryption(Model m) :
  m.getClassDiagramClasses()
   .exists(e|e.implementsInterface("PlainData"))
    ;
cached Boolean requiresSymmetricEncryption(Model m) :
 m.getClassDiagramClasses().attribute.exists
    (e|e.type.name.toString()==encDataSymmName());
cached Boolean requiresSymmKey(Model m) :
 m.getClassDiagramClasses().attribute.exists
    (e|e.type.name.toString()==symmkeyName());
cached Boolean requiresAsymmetricEncryption(Model m) :
 m.getClassDiagramClasses().attribute.exists
    (e|e.type.name.toString()==encDataAsymmName());
cached Boolean requiresPublicKey(Model m) :
 m.getClassDiagramClasses().attribute.exists
    (e|e.type.name.toString()==pubkeyName() );
cached Boolean requiresPrivateKey(Model m) :
 m.getClassDiagramClasses().attribute.exists
    (e|e.type.name.toString()==privkeyName() );

Boolean hasPrivPubKeypair(Model m) :
 m.getClassDiagramClasses().attribute.exists
```

```
    (e|e.type.name.toString()
      ==pubkeyName()) && m.getClassDiagramClasses
        ().attribute.exists(e|e.type.name.toString()
          ==privkeyName());
cached Boolean requiresHashing(Model m) :
  m.getClassDiagramClasses()
  .exists(e|e.implementsInterface(hashDataName()))
  ;
cached Boolean requiresSigning(Model m) :
  m.getClassDiagramClasses()
  .exists(e|e.implementsInterface(signDataName()))
  || m.getClassDiagramClasses().attribute.exists
    (e|e.type.name == signedDataName())
  ;

Boolean requiresMAC(Model m) :
  m.getClassDiagramClasses()
  .attribute.exists(e|e.type.name.toString().matches
    (macDataName()));

Boolean isCard(Class this) :
 this.hasStereotype
   ("SecureMDD::Smartcard") || this.getGeneralClasses
     ().exists(e|e.hasStereotype
       ("SecureMDD::Smartcard"));

Boolean isTerminal(Class this) :
 let res = this.hasStereotype
   ("SecureMDD::Terminal") ||
     this.getGeneralClasses().exists(e|e.hasStereotype
       ("SecureMDD::Terminal")) :
     res;

Boolean isServer(Class this) :
 this.hasStereotype
   ("SecureMDD::Service") || this.getGeneralClasses
     ().exists(e|e.hasStereotype
       ("SecureMDD::Service"));

Boolean isServer(Element this) :
 this.hasStereotype("SecureMDD::Service");

Boolean isSmartcard(Element this) :
 this.hasStereotype("SecureMDD::Smartcard");

Boolean isTerminal(Element this) :
 this.hasStereotype("SecureMDD::Terminal");
cached Boolean hasManualClasses(Model m) :
 m.getClassDiagramClasses().exists(e|e.hasStereotype
   ("SecureMDD::Manual"));
cached Boolean hasManualMethods(Model m) :
 m.getClassDiagramClasses().ownedElement.exists
   (e|e.hasStereotype("SecureMDD::Manual"));
cached Boolean hasServer(Model m) :
```

```
 m.getClassDiagramClasses().exists(e|e.hasStereotype
    ("SecureMDD::Service"));
cached Boolean hasUserMessage(Model m) :
 m.getClassDiagramClasses().exists(e|e.hasStereotype
    ("SecureMDD::Usermessage"));

Boolean isAbstract(Element this) :
 ((Class)this).isAbstract;

Boolean isAbstractOp(Element this) :
 ((Operation)this).isAbstract;

Boolean isStaticOp(Element this) :
 ((Operation)this).isStatic;

Boolean hasInitValue(Property this) :
 !((this.type.metaType
   == Class) && ((Class)(this.type)).isAbstract());

Boolean hasCardSubclasses(Class this) :
 this.getModel().getClassDiagramClasses().select
   (e|e.hasStereotype
     ("SecureMDD::Smartcard")).getGeneralClasses
       ().contains(this) ;

Boolean hasTerminalSubclasses(Class this) :
 this.getModel().getClassDiagramClasses().select
   (e|e.hasStereotype
     ("SecureMDD::Terminal")).getGeneralClasses
       ().contains(this);

Boolean hasServerSubclasses(Class this) :
 this.getModel().getClassDiagramClasses().select
   (e|e.hasStereotype
     ("SecureMDD::Service")).getGeneralClasses
       ().contains(this);

Boolean isMergeNode(DecisionNode this) :
 this.outgoing.size<=1;

Boolean isStructuredActivityEnd
  (ActivityParameterNode this) :
 this.outgoing.size==0;
Boolean isStringType
  (Property this) : this.translateType() == "String";
Boolean isPrimitiveType
  (Property this) : this.translateType
    ().isPrimitiveType();
Boolean isPrimitiveType
  (Parameter this) : this.translateType
    ().isPrimitiveType();
Boolean hasEqualMethod(Property this) :
 let type = this.translateType():
  if type == "String" then true
  else if !type.isPrimitiveType() then true
  else false;
```

```
Boolean isPrimitiveType(String this) :
 targetCard()
  ? (let tn = this :
   (tn=="byte" || tn=="short" || tn=="boolean"))
  : (let tn = this :
   (tn == "String" || tn=="byte" || tn=="int" || tn
     =="boolean"))
 ;

Boolean hasMethods(Class c) :
 ! c.ownedOperation.select(e| e.name.toString
   ().toFirstLower()!= c.name.toString().toFirstLower
     ()).isEmpty;

Boolean hasManualMethods(Class c) :
 ! c.ownedOperation.union(c.general.typeSelect
   (Class).ownedOperation).select(e| e.hasStereotype
     ("SecureMDD::Manual")).isEmpty;

Boolean isPrimitiveTranslatedType
  (String translatedType) :
 (translatedType=="byte" || translatedType
   =="short" || translatedType=="boolean");

Boolean isPrimitiveTypeArray(Property this) :
 targetCard()
  ? (let tn = this.translateType() :
   (tn == "String[]" || tn=="byte[]" || tn=="short[]"))
  : (let tn = this.translateType() :
   (tn.contains("[]")))
 ;

Boolean isEnum(Classifier this) :
 this.metaType==Enumeration;
Boolean isList(Property p) : p.getUpper() != 1;

Boolean hasStereotype(Element this,String sn) :
 this.getAppliedStereotype(sn)!=null;

Boolean hasInitializeStereotype(Element this) :
 this.hasStereotype("SecureMDD::Initialize");

Boolean hasExceptionHandler(ActivityNode node) :
 node.metaType
   == CallBehaviorAction && !(
     (CallBehaviorAction)node).handler.isEmpty;

Boolean hasStaticAttributes(Class this) :
 this.isConstants();

Boolean isConstants(Class this) :
 this.hasStereotype("SecureMDD::Constant");
```

```
Boolean isMessage(Class this) :
 this.hasStereotype("SecureMDD::Message") ||
 this.general.typeSelect(Class).exists(c|c.isMessage
   ());

Boolean isHashData(Class this) :
 this.implementsInterface(hashDataName());

Boolean isHashedData(Class this) :
 this.name.toString()==hashedDataName();

Boolean isEncData(Class this) :
 this.name.toString()==encDataName();

Boolean isEncDataSymm(Class this) :
 this.name.toString()==encDataSymmName();

Boolean isEncDataAsymm(Class this) :
 this.name.toString()==encDataAsymmName();

Boolean isSignedData(Class this) :
 this.name.toString()==signedDataName();

Boolean isPlainData(Class this) :
 this.implementsInterface(plainDataName()) ;

Boolean isSignData(Class this) :
 this.implementsInterface(signDataName());

Boolean isStateAttribute(Property a) :
 a.hasStereotype("SecureMDD::status");

Boolean isSecurityDatatype(Class this) :
 this.package.hasStereotype
   ("SecureMDD::SecurityDatatypes");

Boolean isSecurityDatatype(Type this) :
 (this.metaType
   == Class && ((Class) this).isSecurityDatatype());
Boolean isNonce(Class this) : this.name.toString()
  ==nonceName();
Boolean isNonce(Property this) : this.type
  == this.getModel().getNonce();
Boolean isSecret(Class this) : this.name.toString()
  ==secretName();
Boolean isKey(Class this) : this.name.toString()
  ==keyName();
Boolean isSymmkey(Class this) : this.name.toString()
  ==symmkeyName();
Boolean isAsymmkey(Class this) : this.name.toString()
  ==asymmkeyName();
Boolean isPrivateKey(Class this) : this.name.toString
  ()==privkeyName();
Boolean isPublicKey(Class this) : this.name.toString
```

```
  ()==pubkeyName();
Boolean isMACData(Class this) : this.name.toString()
  ==macDataName();

Boolean isCryptoClass(Class c) :
  c.isPlainData() || c.isEncData() || c.isKey
    () || c.isEncDataSymm() || c.isSymmkey()
  || c.isEncDataAsymm() || c.isPrivateKey
    () || c.isPublicKey() || c.isHashData()
  || c.isHashedData() || c.isSignData
    () || c.isSignedData()
  || c.isNonce() || c.isSecret() || c.isMACData();

Boolean isStateful(Class server):
 let stereotype=server.getAppliedStereotype
   ("SecureMDD::Service"):
 stereotype
   ==null
     ? server.isGeneralStateful() :
       (let b= server.getValue(stereotype,"stateful"):
 b==true ? true : server.isGeneralStateful());

Boolean isGeneralStateful(Class server):
 let gStereotype=server.getGeneralServer
   ().getAppliedStereotype("SecureMDD::Service"):
 gStereotype
   ==null
     ? false : (let b= server.getGeneralServer
       ().getValue(gStereotype,"stateful"):
 b==true ? true : false);

Boolean isStateful(Node server):
 let stereotype=server.getAppliedStereotype
   ("SecureMDD::Service"):
 stereotype
   ==null
     ? server.isGeneralStateful() :
       (let b= server.getValue(stereotype,"stateful"):
 b==true ? true : server.isGeneralStateful());

Boolean isGeneralStateful(Node server):
 let gStereotype=server.getGeneralServer
   ().getAppliedStereotype("SecureMDD::Service"):
 gStereotype
   ==null
     ? false : (let b= server.getGeneralServer
       ().getValue(gStereotype,"stateful"):
 b==true ? true : false);

Boolean isTLSServerSideAuth(Association ass):
 let stereotype=ass.getAppliedStereotype
   ("SecureMDD::TLS"):
 stereotype
   ==null
```

```
      ? false : (let b= ass.getValue
        (stereotype,"ServerSideAuthentication"):
 b==true ? true : false);

Boolean isOperation(Model mo,String s):
 if(mo.getClassDiagramClasses().instanceOperations
   ().select(e|e.name==s).name.first()==s)then
 true
 else false
 ;
cached Boolean hasSeveralInstances(Node component):
 let stereotype=component.getAppliedStereotype
   ("SecureMDD::Service"):
 stereotype
   ==null
     ? true : (let b= ((Integer)component.getValue
       (stereotype,"maxInstances")):
 b==0 || b>1? true : false);
cached Boolean hasSeveralInstances(Class component):
 component.getNode().hasSeveralInstances();
cached Boolean callSeveralInstancesWithSameComponentType
  (Node invoker):
 invoker.attribute.exists(a|a.upper>1 || a.upper==-1);
cached Boolean callSeveralInstancesWithSameComponentType
  (Class invoker):
 invoker.getNode
   ().callSeveralInstancesWithSameComponentType();
cached Boolean callSeveralServiceInstancesWithSameComponentType
  (Node invoker):
 invoker.attribute.exists(a|(a.upper>1 || a.upper
   ==-1) && a.type.isServer());
cached Boolean callSeveralServiceInstancesWithSameComponentType
  (Class invoker):
 invoker.getNode
   ().callSeveralServiceInstancesWithSameComponentType
     ();
cached Boolean callSeveralInstancesFromThisComponent
  (Node invoker, Node component):
 invoker.attribute.exists(a|a.type.name
   ==component.name && (a.upper>1 || a.upper==-1));
cached Boolean callSeveralInstancesFromThisComponent
  (Class invoker, Node component):
 invoker.getNode().attribute.exists(a|a.type.name
   ==component.name && (a.upper>1 || a.upper==-1));

Boolean isStatefulManager(Class manager):
 manager.name.endsWith("StatefulManager");

Boolean isStatefulManager(Node manager):
 manager.name.endsWith("StatefulManager");

Boolean sentBy(Message m, Class termcard) :
 (
   (MessageOccurrenceSpecification)m.sendEvent).covered.exists
```

```
      (l|l.represents.type.name == termcard.name
  || termcard.getModel().getTerminals().addAll
    (termcard.getModel().getCards()).selectFirst
      (e|e.name
        == termcard.name).getGeneralClasses().exists
          (e|e.name == l.represents.type.name));

Boolean isReceivedBy(Message m, Class termcard) :
 (
   (MessageOccurrenceSpecification)m.receiveEvent).covered.exists
     (l|l.represents.type.name == termcard.name
  || termcard.getModel().getTerminals().addAll
    (termcard.getModel().getCards()).selectFirst
      (e|e.name
        == termcard.name).getGeneralClasses().exists
          (e|e.name == l.represents.type.name));

Boolean isOperation(Activity a):
 !a.partition.isEmpty && (!a.ownedElement.typeSelect
   (ActivityParameterNode).isEmpty || !a.ownedElement.typeSelect
     (InitialNode).isEmpty);

Boolean isServiceOperation(Activity a):
 a.isOperation() && a.getModel().getAllActivities
   ().ownedElement.typeSelect
     (CallBehaviorAction).exists(e|e.behavior.name
       == a.name && !e.ownedElement.typeSelect
         (InputPin).incoming.isEmpty);

Boolean isInputParameterNode(ActivityNode a):
 a.metaType
   ==ActivityParameterNode && ((
     (ActivityParameterNode)a).parameter.direction.toString
       ()
         =="in" || (
           (ActivityParameterNode)a).parameter.direction.toString
             ()=="inout");

Boolean isReturnParameterNode(ActivityNode a):
 a.metaType
   ==ActivityParameterNode && ((
     (ActivityParameterNode)a).parameter.direction.toString
       ()=="return");
```

3.3.7 Operations for type translations

```
import uml;
extension psm_templates_shared::names;
extension psm_templates_shared::tests;
extension psm_templates_shared::getter;
extension psm_templates_shared::parse;

String translateTypeForCodingMethod(Property this) :
 targetCard()
```

```
 ? (let t=
  this.translateType() : (
   t=="byte[]" ? "ByteArray" :
   (t=="short" ? "Int" :
   (t=="byte" ? "Byte" :
   (t=="boolean" ? "Boolean" :
   t))))
  )
 : (let t=this.translateType() : (
  t=="byte[]" ? "ByteArray" :
  (t=="int" ? "Int" :
  (t=="byte" ? "Byte" :
  (t=="boolean" ? "Boolean" :
  t)))))
 ;
String toCleanType(Property this) :
 (let typename=this.translateType() :
  (typename=="byte[]" ? "byte" :
  (typename)))
 ;
String translatePrimitiveType(String this) : this;
String translateType(Property this) :
 (this.type.metaType == Enumeration ? enumTypeName() :
 (let typename = this.type.name.toString() :
 (this.upper != 1 ? (typename.isPrimitiveType()
   ? typename+"[]" : "ListOf"+typename) : typename)
  ));
String translateType(Parameter this) :
 (this.type.metaType == Enumeration ? enumTypeName() :
 (let typename = this.type.name.toString() :
 (this.upper != 1 ? (typename.isPrimitiveType()
   ? typename+"[]" : "ListOf"+typename) : typename)
  ));

String translateReturnType(Parameter this) :
 (this == null || this.type.name.toString()
   == "")?("void"):(this.translateType());

String initialValue(Property this) :
 let typename=this.translateType() :
 (!this.getModel().getAllEnumerations().contains
   (this.type) ?
 (typename
   =="boolean"
     ? (this.defaultValue.stringValue
       () != null && this.defaultValue.stringValue
         () != ""
           ? this.defaultValue.stringValue
             () : "false") :
 (typename
   =="int"
     ? (this.defaultValue.stringValue
       () != null && this.defaultValue.stringValue
         () != ""
```

```
            ? this.defaultValue.stringValue() : "0") :
 (typename
   =="short"
     ? (this.defaultValue.stringValue
       () != null && this.defaultValue.stringValue
         () != ""
           ? this.defaultValue.stringValue() : "0") :
 (typename
   =="byte"
     ? (this.defaultValue.stringValue
       () != null && this.defaultValue.stringValue
         () != ""
           ? this.defaultValue.stringValue() : "0") :
 (typename=="String" ? "new String()" :
 (typename
   =="byte[]" ? "new byte[Store.LENGTHOFSTRING]" :
 ( ((this.type.metaType
   == Class) && ((Class)this.type).isListClass() )
     ? (((Class)this.type).getListUpper()
       ==-1
         ? "new "+typename+"()": "new "+typename+"(
           (short)"+(((Class)this.type).getListUpper
             ())+")" ):
"new "+typename+"()")))))) :
  this.type.name.toString()+"."+(
  this.defaultValue.stringValue() != null
    ? this.defaultValue.stringValue().toString() :
   (
   ((Enumeration)this.type).ownedLiteral.exists
     (e|e.name.contains("IDLE"))
       ? (
         (Enumeration)this.type).ownedLiteral.selectFirst
           (e|e.name.contains("IDLE")).name.toString
             () :
  ((Enumeration)this.type).ownedLiteral.last
    ().name.toString()))
 );

String getterName(Property a) :
 (let nam = a.name.toString() :
 ( nam.length > 0 ? "get" + nam.toFirstUpper() : nam)
 );

String setterName(Property a) :
 (let nam = a.name.toString() :
 ( nam.length > 0 ? "set" + nam.toFirstUpper() : nam)
 );
```

Bibliography

[1] M. Balser, W. Reif, G. Schellhorn, K. Stenzel, and A. Thums. Formal system development with KIV. In *Fundamental Approaches to Software Engineering*. Springer LNCS 1783, 2000.

[2] E. Börger and R. F. Stärk. *Abstract State Machines—A Method for High-Level System Design and Analysis*. Springer-Verlag, 2003.

[3] *Java Card 2.2.2 Application Programming Interfaces*, 2006. URL:http://www.oracle.com/technetwork/java/javacard/specs-138637.html.

[4] K. Katkalov, N. Moebius, K. Stenzel, M. Borek, and W. Reif. Model-Driven Testing of Security Protocols with SecureMDD. In *Fifth IFIP International Conference on New Technologies, Mobility and Security (NTMS 2012)*. IEEE XPlore, 2012.

[5] N. Moebius, K. Stenzel, H. Grandy, and W. Reif. SecureMDD: A Model-Driven Development Method for Secure Smart Card Applications. In *Workshop on Secure Software Engineering, SecSE, at ARES 2009*. IEEE Press, 2009.

[6] N. Moebius, K. Stenzel, and W. Reif. Formal Verification of Application-Specific Security Properties in a Model-Driven Approach. In *Proc. of International Symposium on Engineering Secure Software and Systems 2010*. Springer LNCS 5965, 2010.

[7] K. Stenzel, N. Moebius, and W. Reif. Formal verification of QVT transformations for code generation. In *Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011*. Springer LNCS 6981, 2011.